
Amazon CloudWatch

User Guide



Amazon CloudWatch: User Guide

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon CloudWatch?	1
Accessing CloudWatch	1
Related AWS services	1
How CloudWatch works	2
Concepts	2
Namespaces	3
Metrics	3
Dimensions	4
Resolution	5
Statistics	5
Units	6
Periods	6
Aggregation	7
Percentiles	7
Alarms	8
Resources	8
Getting set up	10
Sign up for Amazon Web Services (AWS)	10
Sign in to the Amazon CloudWatch console	10
Set up the AWS CLI	10
Getting started	11
See key metrics from all AWS services	13
Remove a service from appearing in the cross-service dashboard	14
Focus on a single service	15
Focus on a resource group	16
Working with AWS SDKs	17
Using dashboards	18
Create a dashboard	19
Cross-account cross-Region dashboards	19
Creating and using a cross-account cross-Region dashboard with the AWS Management Console	20
Create a cross-account cross-Region dashboard programmatically	21
Creating and working with widgets on CloudWatch dashboards	22
Add or remove a graph	23
Graph metrics manually on a CloudWatch dashboard	25
Edit a graph	25
Add a metrics explorer widget to a CloudWatch dashboard	30
Add an alarm widget to a CloudWatch dashboard	32
Add a custom widget to a CloudWatch dashboard	32
Add or remove a number widget from a CloudWatch dashboard	40
Add or remove a text widget	40
Link and unlink graphs	41
Sharing dashboards	41
Permissions required to share a dashboard	42
Permissions that are granted to people who you share the dashboard with	43
Share a single dashboard with specific users	43
Share a single dashboard publicly	44
Share all CloudWatch dashboards in the account by using SSO	45
Set up SSO for CloudWatch dashboard sharing	45
See how many of your dashboards are shared	46
See which of your dashboards are shared	46
Stop sharing one or more dashboards	46
Review shared dashboard permissions and change permission scope	47
Allowing people that you share with to see composite alarms	48

Allowing people that you share with to see logs table widgets	49
Allowing people that you share with to see custom widgets	50
Use live data	51
Viewing an animated dashboard	51
Add a dashboard to your Favorites list	52
Change the period override setting or refresh interval	52
Change the time range or time zone format	53
Using metrics	55
Query your metrics with CloudWatch Metrics Insights	55
Build your queries	56
Query components and syntax	56
Using Metrics Insights queries with metric math	62
SQL inference	62
Sample queries	63
Metrics Insights limits	69
Metrics Insights glossary	70
Troubleshooting Metrics Insights	70
Viewing available metrics	71
Searching for available metrics	74
Getting statistics for a metric	75
CloudWatch statistics definitions	75
Getting statistics for a specific resource	78
Aggregating statistics across resources	81
Aggregating statistics by Auto Scaling group	82
Aggregating statistics by AMI	84
Graphing metrics	85
Graphing a metric	85
Using dynamic labels	88
Modifying the time range or time zone format for a graph	90
Modifying the y-axis for a graph	93
Creating an alarm from a metric on a graph	93
Publishing custom metrics	94
High-resolution metrics	95
Using dimensions	95
Publishing single data points	96
Publishing statistic sets	97
Publishing the value zero	97
Using metric math	97
Adding a math expression to a CloudWatch graph	97
Metric math syntax and functions	98
Using IF expressions	116
Anomaly detection on metric math	119
Using search expressions in graphs	119
Search expression syntax	120
Search expression examples	124
Creating a graph with a search expression	125
Use metrics explorer to monitor resources by their tags and properties	128
CloudWatch agent configuration for metrics explorer	129
Using alarms	130
Metric alarm states	130
Evaluating an alarm	131
Alarm actions	132
Configuring how alarms treat missing data	132
How alarm state is evaluated when data is missing	133
High-resolution alarms	135
Alarms on math expressions	135
Percentile-based alarms and low data samples	136

CloudWatch alarms and Amazon EventBridge	136
Common features of CloudWatch alarms	136
Setting up an SNS topic	137
Setting up an Amazon SNS topic using the AWS Management Console	137
Setting up an SNS topic using the AWS CLI	138
Create an alarm based on a static threshold	139
Creating an alarm based on anomaly detection	141
Modifying an anomaly detection model	143
Deleting an anomaly detection model	143
Creating an alarm based on a metric math expression	144
Creating a composite alarm	146
Editing or deleting a CloudWatch alarm	148
Creating a CPU usage alarm	149
Setting up a CPU usage alarm using the AWS Management Console	149
Setting up a CPU usage alarm using the AWS CLI	150
Creating a load balancer latency alarm	151
Setting up a latency alarm using the AWS Management Console	151
Setting up a latency alarm using the AWS CLI	152
Creating a storage throughput alarm	152
Setting up a storage throughput alarm using the AWS Management Console	152
Setting up a storage throughput alarm using the AWS CLI	153
Create alarms to stop, terminate, reboot, or recover an EC2 instance	154
Adding stop actions to Amazon CloudWatch alarms	155
Adding terminate actions to Amazon CloudWatch alarms	156
Adding reboot actions to Amazon CloudWatch alarms	156
Adding recover actions to Amazon CloudWatch alarms	157
Viewing the history of triggered alarms and actions	159
Creating a billing alarm	159
Enabling billing alerts	159
Creating a billing alarm	160
Deleting a billing alarm	161
Hiding Amazon EC2 Auto Scaling alarms	161
Using synthetic monitoring	162
Required roles and permissions	163
Required roles and permissions for users who manage CloudWatch canaries	163
Required roles and permissions for canaries	164
AWS managed policies for CloudWatch Synthetics	168
Limiting a user to viewing specific canaries	172
Creating a canary	173
Resources that are created for canaries	175
Using canary blueprints	175
Using the CloudWatch Synthetics Recorder for Google Chrome	179
Synthetics runtime versions	181
Writing a canary script	192
Library functions available for canary scripts	200
Scheduling canary runs using cron	230
Troubleshooting a failed canary	232
Canary runtime version upgrade and downgrade issues	232
Waiting for an element to appear	233
Node is either not visible or not an HTMLElement for page.click()	233
Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied ..	233
Error: Protocol error (Runtime.callFunctionOn): Target closed.	233
Canary Failed. Error: No datapoint - Canary Shows timeout error	234
Trying to access an internal endpoint	234
Cross-origin request sharing (CORS) issue	234
Troubleshooting a canary on a VPC	235
Sample code for canary scripts	235

Samples for Node.js and Puppeteer	235
Samples for Python and Selenium	239
Canaries and X-Ray tracing	239
Running a canary on a VPC	240
Giving internet access to your canary on a VPC	240
Encrypting canary artifacts	241
Updating artifact location and encryption when using visual monitoring	242
Viewing canary statistics and details	243
CloudWatch metrics published by canaries	244
Editing or deleting a canary	246
Monitoring canary events with Amazon EventBridge	247
Example events from CloudWatch Synthetics	247
Prerequisites for creating EventBridge rules	249
Create an EventBridge rule (CLI)	249
Using metric streams	251
Setting up a metric stream	251
Setting up a metric stream to an AWS service (data lake scenario)	252
Setting up a metric stream to a third-party solution	254
Metric stream operation and maintenance	254
Monitoring your metric streams with CloudWatch metrics	255
Trust between CloudWatch and Kinesis Data Firehose	256
Metric streams output formats	256
JSON format	256
OpenTelemetry 0.7.0 format	258
Troubleshooting	266
Using ServiceLens to monitor the health of your applications	268
Deploying ServiceLens	269
Deploying AWS X-Ray	269
Deploying the CloudWatch agent and the X-Ray daemon	271
Using the service map	277
Using the traces view	279
Using the resource health view	279
Prerequisites	280
ServiceLens troubleshooting	281
I don't see all my logs	281
I don't see all my alarms on the service map	282
I don't see some AWS resources on the service map	282
There are too many nodes on my service map	283
Cross-account cross-Region CloudWatch console	284
Enabling cross-account cross-Region functionality	284
(Optional) Integrate with AWS Organizations	287
Troubleshooting	287
Disabling and cleaning up after using cross-account	288
Using anomaly detection	289
How anomaly detection works	291
Anomaly detection on metric math	291
Using Contributor Insights to analyze high-cardinality data	293
Creating a Contributor Insights rule	293
Contributor Insights rule syntax	296
Example rules	299
Viewing Contributor Insights reports	301
Graphing metrics generated by rules	302
Setting an alarm on Contributor Insights metric data	303
Using Contributor Insights built-in rules	304
Using Container Insights	305
Supported platforms	305
CloudWatch agent container image	306

Supported Regions	306
Setting up Container Insights	307
Setting up Container Insights on Amazon ECS	307
Setting up Container Insights on Amazon EKS and Kubernetes	320
Viewing Container Insights metrics	344
Viewing the top contributors	344
Using CloudWatch Logs Insights to view Container Insights data	345
Use case: Seeing task-level metrics in Amazon ECS containers	346
Other sample queries for Container Insights	346
Metrics collected by Container Insights	347
Amazon ECS Container Insights metrics	347
Amazon EKS and Kubernetes Container Insights metrics	351
Performance log reference	354
Performance log events for Amazon ECS	354
Performance log events for Amazon EKS and Kubernetes	357
Relevant fields in performance log events for Amazon EKS and Kubernetes	368
Container Insights Prometheus metrics monitoring	374
Set up and configure on Amazon ECS clusters	375
Set up and configure on Amazon EKS and Kubernetes clusters	414
Integration with Application Insights	454
Troubleshooting Container Insights	454
Failed deployment on Amazon EKS or Kubernetes	454
Unauthorized panic: Cannot retrieve cadvisor data from kubelet	454
Deploying Container Insights on a deleted and re-created cluster on Amazon ECS	455
Invalid endpoint error	455
Metrics don't appear in the console	455
Pod metrics missing on Amazon EKS or Kubernetes after upgrading cluster	455
No pod metrics when using Bottlerocket for Amazon EKS	456
No container filesystem metrics when using the containerd runtime for Amazon EKS or Kubernetes	456
Unexpected log volume increase from CloudWatch agent when collecting Prometheus metrics ..	456
Latest docker image mentioned in release notes not found from Dockerhub	457
CrashLoopBackoff error on the CloudWatch agent	457
CloudWatch agent or FluentD pod stuck in pending	457
Building your own CloudWatch agent Docker image	457
Deploying other CloudWatch agent features in your containers	457
Using Lambda Insights	459
Getting started with Lambda Insights	459
Available versions of the Lambda Insights extension	459
Using the console to enable Lambda Insights on an existing Lambda function	469
Using the AWS CLI to enable Lambda Insights on an existing Lambda function	469
Using the AWS SAM CLI to enable Lambda Insights on an existing Lambda function	470
Using AWS CloudFormation to enable Lambda Insights on an existing Lambda function	471
Using the AWS CDK to enable Lambda Insights on an existing Lambda function	472
Using Serverless Framework to enable Lambda Insights on an existing Lambda function	473
Enabling Lambda Insights on a Lambda container image deployment	474
Viewing your Lambda Insights metrics	476
Metrics collected by Lambda Insights	476
Troubleshooting and known issues	479
I don't see any metrics from Lambda Insights	479
Known issues	480
Example telemetry event	480
Collect metrics and logs with the CloudWatch agent	482
Installing the CloudWatch agent	483
Installing the CloudWatch agent using the command line	484
Install the CloudWatch agent using Systems Manager	498
Installing the CloudWatch agent on new instances using AWS CloudFormation	511

Verifying the signature of the CloudWatch agent package	515
Create the CloudWatch agent configuration file	522
Create the CloudWatch agent configuration file with the wizard	522
Manually create or edit the CloudWatch agent configuration file	527
Metrics collected by the CloudWatch agent	574
Metrics collected by the CloudWatch agent on Windows Server instances	575
Metrics collected by the CloudWatch agent on Linux and macOS instances	575
OpenTelemetry support in the CloudWatch agent	584
IAM permissions	585
CloudWatch agent OpenTelemetry configuration	586
Use the command line to manage the CloudWatch agent with OpenTelemetry support	586
Use Systems Manager to manage the CloudWatch agent with the embedded OpenTelemetry Collector	587
Generating OpenTelemetry metrics and traces	589
Common scenarios with the CloudWatch agent	589
Running the CloudWatch agent as a different user	589
Adding custom dimensions to metrics collected by the CloudWatch agent	591
Multiple CloudWatch agent configuration files	591
Aggregating or rolling up metrics collected by the CloudWatch agent	593
Collecting high-resolution metrics with the CloudWatch agent	594
Sending metrics and logs to a different account	594
Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent	596
Troubleshooting the CloudWatch agent	596
CloudWatch agent command line parameters	597
Installing the CloudWatch agent using Run Command fails	597
The CloudWatch agent won't start	597
Verify that the CloudWatch agent is running	597
The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region	598
The CloudWatch agent won't start on Windows Server	598
Unable to find credentials on Windows Server	599
Where are the metrics?	599
I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console	599
CloudWatch agent files and locations	599
Finding information about CloudWatch agent versions	600
Logs generated by the CloudWatch agent	601
Stopping and restarting the CloudWatch agent	601
Detect common application problems with CloudWatch Application Insights	603
What is Amazon CloudWatch Application Insights?	603
Features	604
Concepts	604
Pricing	605
Related services	605
Supported application components	607
Supported technology stacks	608
How Application Insights works	608
How Application Insights monitors applications	608
Data retention	609
Quotas	609
SSM packages used by Application Insights	609
Get started	615
Access CloudWatch Application Insights	615
Prerequisites	615
IAM policy	616
Permissions for account-based application onboarding	617
Set up, configure, and manage your application	617

Work with component configurations	634
Template fragment	634
Sections	635
Example configurations for relevant services	640
Use CloudFormation templates	668
Create an Application Insights application for the entire AWS CloudFormation stack	668
Create an Application Insights application with detailed settings	670
Create an Application Insights application with CUSTOM mode component configuration	672
Create an Application Insights application with DEFAULT mode component configuration	674
Create an Application Insights application with DEFAULT_WITH_OVERWRITE mode component configuration	675
Tutorial: Set up monitoring for SAP HANA	677
Supported environments	677
Supported operating systems	677
Features	678
Prerequisites	678
Set up monitoring	679
Manage monitoring	680
Configure the alarm threshold	680
Troubleshooting problems detected	681
Anomaly detection	683
Troubleshooting Application Insights	684
Tutorial: Set up monitoring for .NET and SQL	685
Use case scenario	685
Prerequisites	686
Deploy resources	686
Set up monitoring with Amazon CloudWatch Application Insights	687
Simulate problem scenarios and view insights	688
View and troubleshoot Application Insights	690
CloudWatch console overview	690
Application Insights problem summary page	690
Feedback	691
Configuration errors	691
Supported logs and metrics	692
Amazon Elastic Compute Cloud (EC2)	694
Elastic Block Store (EBS)	700
Elastic Load Balancer (ELB)	701
Application ELB	701
Amazon EC2 Auto Scaling groups	701
Amazon Simple Queue Server (SQS)	702
Amazon Relational Database Service (RDS)	703
AWS Lambda function	705
Amazon DynamoDB table	705
Amazon S3 bucket	705
AWS Step Functions	706
API Gateway REST API stages	708
SAP HANA	708
HA Cluster	712
Java	713
Amazon Elastic Container Service (Amazon ECS)	713
Kubernetes on AWS	715
Amazon FSx	717
Metrics with data points requirements	717
Recommended metrics	725
Performance Counter metrics	740
Services that publish CloudWatch metrics	746
Alarm events and EventBridge	753

Sample events from CloudWatch	753
Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format	761
Generating logs using the embedded metric format	761
Using the client libraries	762
Using the embedded metric format with AWS Distro for OpenTelemetry	762
Manually generating embedded metric format logs	762
Viewing your metrics and logs in the console	774
Use CloudWatch RUM	776
IAM policies to use CloudWatch RUM	777
Set up an application to use CloudWatch RUM	778
Step 1: Authorize your application to send data to AWS	778
Step 2: Create an app monitor	780
(Optional) Step 3: Manually modify the code snippet to configure the CloudWatch RUM web client	781
Step 4: Insert the code snippet into your application	783
Step 5: Test your app monitor setup by generating user events	783
Configuring the CloudWatch RUM web client with CDN installation	784
Arguments	784
Configuration options	784
Viewing the CloudWatch RUM dashboard	785
How CloudWatch RUM sets Apdex scores	786
CloudWatch metrics that you can collect with CloudWatch RUM	786
Data protection and data privacy with CloudWatch RUM	789
CloudWatch RUM web client cookies	789
Information collected by the CloudWatch RUM web client	790
RUM event schema	790
RUM event metadata	790
RUM event details	791
Manage your applications that use CloudWatch RUM	806
How do I find a code snippet that I've already generated?	806
Edit your application	806
Stop using CloudWatch RUM or delete an app monitor	806
CloudWatch RUM quotas	807
Troubleshooting	807
There is no data for my application	807
Data has stopped being recorded for my application	807
Perform launches and A/B experiments with CloudWatch Evidently	808
IAM policies to use Evidently	809
Create projects, features, launches, and experiments	810
Create a new project	810
Add a feature to a project	811
Create a launch	812
Create an experiment	814
Manage features, launches, and experiments	816
See the current evaluation rules and audience traffic for a feature	817
Modify launch traffic	817
Modify a launch's future steps	818
Modify experiment traffic	818
Stop a launch	819
Stop an experiment	819
Adding code to your application	819
Project data storage	820
Format of evaluation event logs	820
IAM policy and encryption for evaluation event storage in Amazon S3	821
View launch results in the dashboard	822
View experiment results in the dashboard	822
How CloudWatch Evidently collects and stores data	823

CloudWatch Evidently quotas	824
Tutorial: A/B testing with the Evidently sample application	824
Step 1: Download the sample application	824
Step 2: Add the Evidently endpoint and set up credentials	825
Step 3: Set up code for the feature evaluation	825
Step 4: Set up code for the experiment metrics	826
Step 5: Create the project, feature, and experiment	829
Step 6: Start the experiment and test CloudWatch Evidently	830
AWS usage metrics	832
Visualizing your service quotas and setting alarms	832
AWS API usage metrics	833
CloudWatch usage metrics	838
CloudWatch tutorials	840
Scenario: Monitor estimated charges	840
Step 1: Enable billing alerts	840
Step 2: Create a billing alarm	841
Step 3: Check the alarm status	842
Step 4: Edit a billing alarm	842
Step 5: Delete a billing alarm	842
Scenario: Publish metrics	842
Step 1: Define the data configuration	843
Step 2: Add metrics to CloudWatch	843
Step 3: Get statistics from CloudWatch	844
Step 4: View graphs with the console	844
Tagging your CloudWatch resources	845
Supported resources in CloudWatch	845
Managing tags	845
Tag naming and usage conventions	846
Code examples	847
Actions	847
Create an alarm that watches a metric	848
Delete alarms	853
Describe alarm history	857
Describe alarms for a metric	859
Disable alarm actions	863
Enable alarm actions	867
Get dashboard details	873
Get metric statistics	874
List dashboards	875
List metrics	877
Put a set of data into a metric	882
Put data into a metric	883
Scenarios	887
Manage Amazon CloudWatch metrics and alarms	887
Security	893
Data protection	893
Encryption in transit	894
Identity and access management	894
Authentication	894
Access control	895
CloudWatch dashboard permissions update	896
Overview of managing access	896
Using identity-based policies (IAM policies)	899
Using condition keys to limit access to CloudWatch namespaces	919
Using condition keys to limit Contributor Insights users' access to log groups	920
Using condition keys to limit alarm actions	921
Using service-linked roles	922

Using service-linked roles for CloudWatch RUM	926
Using service-linked roles for Application Insights	930
AWS managed policies for Application Insights	937
Amazon CloudWatch permissions reference	942
Compliance validation	951
Resilience	951
Infrastructure security	951
Network isolation	952
Interface VPC endpoints	952
CloudWatch	952
CloudWatch Synthetics	954
Security considerations for Synthetics canaries	955
Use secure connections	955
Canary naming considerations	956
Secrets in canary code	956
Permissions considerations	956
Stack traces and exception messages	956
Scope your IAM roles narrowly	957
Sensitive data redaction	957
Logging API calls with AWS CloudTrail	958
CloudWatch information in CloudTrail	958
Example: CloudWatch log file entries	959
CloudWatch Synthetics information in CloudTrail	961
Example: CloudWatch Synthetics log file entries	961
Grafana integration	964
Service quotas	965
Document history	969

What is Amazon CloudWatch?

Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications.

The CloudWatch home page automatically displays metrics about every AWS service you use. You can additionally create custom dashboards to display metrics about your custom applications, and display custom collections of metrics that you choose.

You can create alarms that watch metrics and send notifications or automatically make changes to the resources you are monitoring when a threshold is breached. For example, you can monitor the CPU usage and disk reads and writes of your Amazon EC2 instances and then use that data to determine whether you should launch additional instances to handle increased load. You can also use this data to stop under-used instances to save money.

With CloudWatch, you gain system-wide visibility into resource utilization, application performance, and operational health.

Accessing CloudWatch

You can access CloudWatch using any of the following methods:

- **Amazon CloudWatch console** – <https://console.aws.amazon.com/cloudwatch/>
- **AWS CLI** – For more information, see [Getting Set Up with the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).
- **CloudWatch API** – For more information, see the [Amazon CloudWatch API Reference](#).
- **AWS SDKs** – For more information, see [Tools for Amazon Web Services](#).

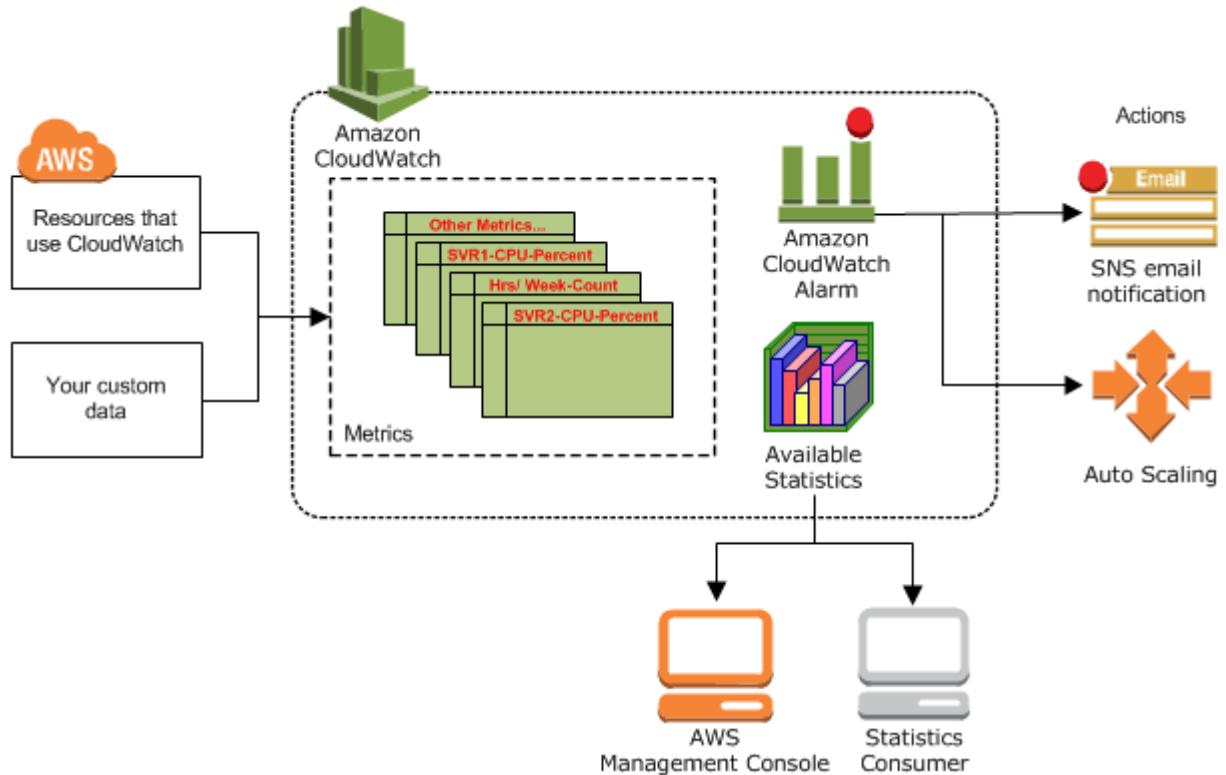
Related AWS services

The following services are used along with Amazon CloudWatch:

- **Amazon Simple Notification Service (Amazon SNS)** coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. You use Amazon SNS with CloudWatch to send messages when an alarm threshold has been reached. For more information, see [Setting up Amazon SNS notifications \(p. 137\)](#).
- **Amazon EC2 Auto Scaling** enables you to automatically launch or terminate Amazon EC2 instances based on user-defined policies, health status checks, and schedules. You can use a CloudWatch alarm with Amazon EC2 Auto Scaling to scale your EC2 instances based on demand. For more information, see [Dynamic Scaling](#) in the [Amazon EC2 Auto Scaling User Guide](#).
- **AWS CloudTrail** enables you to monitor the calls made to the Amazon CloudWatch API for your account, including calls made by the AWS Management Console, AWS CLI, and other services. When CloudTrail logging is turned on, CloudWatch writes log files to the Amazon S3 bucket that you specified when you configured CloudTrail. For more information, see [Logging Amazon CloudWatch API calls with AWS CloudTrail \(p. 958\)](#).
- **AWS Identity and Access Management (IAM)** is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). For more information, see [Identity and access management for Amazon CloudWatch \(p. 894\)](#).

How Amazon CloudWatch works

Amazon CloudWatch is basically a metrics repository. An AWS service—such as Amazon EC2—puts metrics into the repository, and you retrieve statistics based on those metrics. If you put your own custom metrics into the repository, you can retrieve statistics on these metrics as well.



You can use metrics to calculate statistics and then present the data graphically in the CloudWatch console. For more information about the other AWS resources that generate and send metrics to CloudWatch, see [AWS services that publish CloudWatch metrics \(p. 746\)](#).

You can configure alarm actions to stop, start, or terminate an Amazon EC2 instance when certain criteria are met. In addition, you can create alarms that initiate Amazon EC2 Auto Scaling and Amazon Simple Notification Service (Amazon SNS) actions on your behalf. For more information about creating CloudWatch alarms, see [Alarms \(p. 8\)](#).

AWS Cloud computing resources are housed in highly available data center facilities. To provide additional scalability and reliability, each data center facility is located in a specific geographical area, known as a *Region*. Each Region is designed to be completely isolated from the other Regions, to achieve the greatest possible failure isolation and stability. Metrics are stored separately in Regions, but you can use CloudWatch cross-Region functionality to aggregate statistics from different Regions. For more information, see [Cross-account cross-Region CloudWatch console \(p. 284\)](#) and [Regions and Endpoints](#) in the [Amazon Web Services General Reference](#).

Amazon CloudWatch concepts

The following terminology and concepts are central to your understanding and use of Amazon CloudWatch:

- [Namespaces \(p. 3\)](#)

- [Metrics \(p. 3\)](#)
- [Dimensions \(p. 4\)](#)
- [Resolution \(p. 5\)](#)
- [Statistics \(p. 5\)](#)
- [Percentiles \(p. 7\)](#)
- [Alarms \(p. 8\)](#)

Namespaces

A *namespace* is a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

There is no default namespace. You must specify a namespace for each data point you publish to CloudWatch. You can specify a namespace name when you create a metric. These names must contain valid XML characters, and be fewer than 256 characters in length. Possible characters are: alphanumeric characters (0-9A-Za-z), period (.), hyphen (-), underscore (_), forward slash (/), hash (#), and colon (:).

The AWS namespaces typically use the following naming convention: AWS/*service*. For example, Amazon EC2 uses the AWS/EC2 namespace. For the list of AWS namespaces, see [AWS services that publish CloudWatch metrics \(p. 746\)](#).

Metrics

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Think of a metric as a variable to monitor, and the data points as representing the values of that variable over time. For example, the CPU usage of a particular EC2 instance is one metric provided by Amazon EC2. The data points themselves can come from any application or business activity from which you collect data.

By default, many AWS services provide free metrics for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). For a charge, you can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. For custom metrics, you can add the data points in any order, and at any rate you choose. You can retrieve statistics about those data points as an ordered set of time-series data.

Metrics exist only in the Region in which they are created. Metrics cannot be deleted, but they automatically expire after 15 months if no new data is published to them. Data points older than 15 months expire on a rolling basis; as new data points come in, data older than 15 months is dropped.

Metrics are uniquely defined by a name, a namespace, and zero or more dimensions. Each data point in a metric has a time stamp, and (optionally) a unit of measure. You can retrieve statistics from CloudWatch for any metric.

For more information, see [Viewing available metrics \(p. 71\)](#) and [Publishing custom metrics \(p. 94\)](#).

Time stamps

Each metric data point must be associated with a time stamp. The time stamp can be up to two weeks in the past and up to two hours into the future. If you do not provide a time stamp, CloudWatch creates a time stamp for you based on the time the data point was received.

Time stamps are `dateTime` objects, with the complete date plus hours, minutes, and seconds (for example, 2016-10-31T23:59:59Z). For more information, see [dateTime](#). Although it is not required,

we recommend that you use Coordinated Universal Time (UTC). When you retrieve statistics from CloudWatch, all times are in UTC.

CloudWatch alarms check metrics based on the current time in UTC. Custom metrics sent to CloudWatch with time stamps other than the current UTC time can cause alarms to display the **Insufficient Data** state or result in delayed alarms.

Metrics retention

CloudWatch retains metric data as follows:

- Data points with a period of less than 60 seconds are available for 3 hours. These data points are high-resolution custom metrics.
- Data points with a period of 60 seconds (1 minute) are available for 15 days
- Data points with a period of 300 seconds (5 minute) are available for 63 days
- Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)

Data points that are initially published with a shorter period are aggregated together for long-term storage. For example, if you collect data using a period of 1 minute, the data remains available for 15 days with 1-minute resolution. After 15 days this data is still available, but is aggregated and is retrievable only with a resolution of 5 minutes. After 63 days, the data is further aggregated and is available with a resolution of 1 hour.

Note

Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a [list-metrics](#) command. The best way to retrieve these metrics is with the [get-metric-data](#) or [get-metric-statistics](#) commands in the AWS CLI.

Dimensions

A *dimension* is a name/value pair that is part of the identity of a metric. You can assign up to 10 dimensions to a metric.

Every metric has specific characteristics that describe it, and you can think of dimensions as categories for those characteristics. Dimensions help you design a structure for your statistics plan. Because dimensions are part of the unique identifier for a metric, whenever you add a unique name/value pair to one of your metrics, you are creating a new variation of that metric.

AWS services that send data to CloudWatch attach dimensions to each metric. You can use dimensions to filter the results that CloudWatch returns. For example, you can get statistics for a specific EC2 instance by specifying the `InstanceId` dimension when you search for metrics.

For metrics produced by certain AWS services, such as Amazon EC2, CloudWatch can aggregate data across dimensions. For example, if you search for metrics in the `AWS/EC2` namespace but do not specify any dimensions, CloudWatch aggregates all data for the specified metric to create the statistic that you requested. CloudWatch does not aggregate across dimensions for your custom metrics.

Dimension combinations

CloudWatch treats each unique combination of dimensions as a separate metric, even if the metrics have the same metric name. You can only retrieve statistics using combinations of dimensions that you specifically published. When you retrieve statistics, specify the same values for the namespace, metric name, and dimension parameters that were used when the metrics were created. You can also specify the start and end times for CloudWatch to use for aggregation.

For example, suppose that you publish four distinct metrics named ServerStats in the DataCenterMetric namespace with the following properties:

```
Dimensions: Server=Prod, Domain=Frankfurt, Unit: Count, Timestamp: 2016-10-31T12:30:00Z,
Value: 105
Dimensions: Server=Beta, Domain=Frankfurt, Unit: Count, Timestamp: 2016-10-31T12:31:00Z,
Value: 115
Dimensions: Server=Prod, Domain=Rio,           Unit: Count, Timestamp: 2016-10-31T12:32:00Z,
Value: 95
Dimensions: Server=Beta, Domain=Rio,           Unit: Count, Timestamp: 2016-10-31T12:33:00Z,
Value: 97
```

If you publish only those four metrics, you can retrieve statistics for these combinations of dimensions:

- Server=Prod,Domain=Frankfurt
- Server=Prod,Domain=Rio
- Server=Beta,Domain=Frankfurt
- Server=Beta,Domain=Rio

You can't retrieve statistics for the following dimensions or if you specify no dimensions. (The exception is by using the metric math **SEARCH** function, which can retrieve statistics for multiple metrics. For more information, see [Using search expressions in graphs \(p. 119\)](#).)

- Server=Prod
- Server=Beta
- Domain=Frankfurt
- Domain=Rio

Resolution

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every PutMetricData call for a custom metric is charged, so calling PutMetricData more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Statistics

Statistics are metric data aggregations over specified periods of time. CloudWatch provides statistics based on the metric data points provided by your custom data or provided by other AWS services to

CloudWatch. Aggregations are made using the namespace, metric name, dimensions, and the data point unit of measure, within the time period you specify.

For detailed definitions of the statistics supported by CloudWatch, see [CloudWatch statistics definitions \(p. 75\)](#).

Units

Each statistic has a unit of measure. Example units include Bytes, Seconds, Count, and Percent. For the complete list of the units that CloudWatch supports, see the [MetricDatum](#) data type in the [Amazon CloudWatch API Reference](#).

You can specify a unit when you create a custom metric. If you do not specify a unit, CloudWatch uses None as the unit. Units help provide conceptual meaning to your data. Though CloudWatch attaches no significance to a unit internally, other applications can derive semantic information based on the unit.

Metric data points that specify a unit of measure are aggregated separately. When you get statistics without specifying a unit, CloudWatch aggregates all data points of the same unit together. If you have two otherwise identical metrics with different units, two separate data streams are returned, one for each unit.

Periods

A *period* is the length of time associated with a specific Amazon CloudWatch statistic. Each statistic represents an aggregation of the metrics data collected for a specified period of time. Periods are defined in numbers of seconds, and valid values for period are 1, 5, 10, 30, or any multiple of 60. For example, to specify a period of six minutes, use 360 as the period value. You can adjust how the data is aggregated by varying the length of the period. A period can be as short as one second or as long as one day (86,400 seconds). The default value is 60 seconds.

Only custom metrics that you define with a storage resolution of 1 second support sub-minute periods. Even though the option to set a period below 60 is always available in the console, you should select a period that aligns to how the metric is stored. For more information about metrics that support sub-minute periods, see [High-resolution metrics \(p. 95\)](#).

When you retrieve statistics, you can specify a period, start time, and end time. These parameters determine the overall length of time associated with the statistics. The default values for the start time and end time get you the last hour's worth of statistics. The values that you specify for the start time and end time determine how many periods CloudWatch returns. For example, retrieving statistics using the default values for the period, start time, and end time returns an aggregated set of statistics for each minute of the previous hour. If you prefer statistics aggregated in ten-minute blocks, specify a period of 600. For statistics aggregated over the entire hour, specify a period of 3600.

When statistics are aggregated over a period of time, they are stamped with the time corresponding to the beginning of the period. For example, data aggregated from 7:00pm to 8:00pm is stamped as 7:00pm. Additionally, data aggregated between 7:00pm and 8:00pm begins to be visible at 7:00pm, then the values of that aggregated data may change as CloudWatch collects more samples during the period.

Periods are also important for CloudWatch alarms. When you create an alarm to monitor a specific metric, you are asking CloudWatch to compare that metric to the threshold value that you specified. You have extensive control over how CloudWatch makes that comparison. Not only can you specify the period over which the comparison is made, but you can also specify how many evaluation periods are used to arrive at a conclusion. For example, if you specify three evaluation periods, CloudWatch compares a window of three data points. CloudWatch only notifies you if the oldest data point is breaching and the others are breaching or missing. For metrics that are continuously emitted, CloudWatch doesn't notify you until three failures are found.

Aggregation

Amazon CloudWatch aggregates statistics according to the period length that you specify when retrieving statistics. You can publish as many data points as you want with the same or similar time stamps. CloudWatch aggregates them according to the specified period length. CloudWatch does not automatically aggregate data across Regions, but you can use metric math to aggregate metrics from different Regions.

You can publish data points for a metric that share not only the same time stamp, but also the same namespace and dimensions. CloudWatch returns aggregated statistics for those data points. You can also publish multiple data points for the same or different metrics, with any time stamp.

For large datasets, you can insert a pre-aggregated dataset called a *statistic set*. With statistic sets, you give CloudWatch the Min, Max, Sum, and SampleCount for a number of data points. This is commonly used when you need to collect data many times in a minute. For example, suppose you have a metric for the request latency of a webpage. It doesn't make sense to publish data with every webpage hit. We suggest that you collect the latency of all hits to that webpage, aggregate them once a minute, and send that statistic set to CloudWatch.

Amazon CloudWatch doesn't differentiate the source of a metric. If you publish a metric with the same namespace and dimensions from different sources, CloudWatch treats this as a single metric. This can be useful for service metrics in a distributed, scaled system. For example, all the hosts in a web server application could publish identical metrics representing the latency of requests they are processing. CloudWatch treats these as a single metric, allowing you to get the statistics for minimum, maximum, average, and sum of all requests across your application.

Percentiles

A *percentile* indicates the relative standing of a value in a dataset. For example, the 95th percentile means that 95 percent of the data is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.

Percentiles are often used to isolate anomalies. In a normal distribution, 95 percent of the data is within two standard deviations from the mean and 99.7 percent of the data is within three standard deviations from the mean. Any data that falls outside three standard deviations is often considered to be an anomaly because it differs so greatly from the average value. For example, suppose that you are monitoring the CPU utilization of your EC2 instances to ensure that your customers have a good experience. If you monitor the average, this can hide anomalies. If you monitor the maximum, a single anomaly can skew the results. Using percentiles, you can monitor the 95th percentile of CPU utilization to check for instances with an unusually heavy load.

Some CloudWatch metrics support percentiles as a statistic. For these metrics, you can monitor your system and applications using percentiles as you would when using the other CloudWatch statistics (Average, Minimum, Maximum, and Sum). For example, when you create an alarm, you can use percentiles as the statistical function. You can specify the percentile with up to ten decimal places (for example, p95.0123456789).

Percentile statistics are available for custom metrics as long as you publish the raw, unsummarized data points for your custom metric. Percentile statistics are not available for metrics when any of the metric values are negative numbers.

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can only retrieve percentile statistics for this data if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following AWS services include metrics that support percentile statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

CloudWatch also supports trimmed mean and other performance statistics, which can have a similar use as percentiles. For more information, see [CloudWatch statistics definitions \(p. 75\)](#).

Alarms

You can use an *alarm* to automatically initiate actions on your behalf. An alarm watches a single metric over a specified time period, and performs one or more specified actions, based on the value of the metric relative to a threshold over time. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy. You can also add alarms to dashboards.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

When creating an alarm, select an alarm monitoring period that is greater than or equal to the metric's resolution. For example, basic monitoring for Amazon EC2 provides metrics for your instances every 5 minutes. When setting an alarm on a basic monitoring metric, select a period of at least 300 seconds (5 minutes). Detailed monitoring for Amazon EC2 provides metrics for your instances with a resolution of 1 minute. When setting an alarm on a detailed monitoring metric, select a period of at least 60 seconds (1 minute).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see [Publishing custom metrics \(p. 94\)](#).

For more information, see [Using Amazon CloudWatch alarms \(p. 130\)](#) and [Creating an alarm from a metric on a graph \(p. 93\)](#).

Amazon CloudWatch resources

The following related resources can help you as you work with this service.

Resource	Description
Amazon CloudWatch FAQs	The FAQ covers the top questions developers have asked about this product.
AWS Developer Center	A central starting point to find documentation, code examples, release notes, and other information to help you build innovative applications with AWS.
AWS Management Console	The console allows you to perform most of the functions of Amazon CloudWatch and various other AWS offerings without programming.

Resource	Description
Amazon CloudWatch Discussion Forums	Community-based forum for developers to discuss technical questions related to Amazon CloudWatch.
AWS Support	The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
Amazon CloudWatch product information	The primary webpage for information about Amazon CloudWatch.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse, etc.

Getting set up

To use Amazon CloudWatch you need an AWS account. Your AWS account allows you to use services (for example, Amazon EC2) to generate metrics that you can view in the CloudWatch console, a point-and-click web-based interface. In addition, you can install and configure the AWS command line interface (CLI).

Sign up for Amazon Web Services (AWS)

When you create an AWS account, we automatically sign up your account for all AWS services. You pay only for the services that you use.

If you have an account already, skip to the next step. If you don't have an account, use the following procedure to create one.

To sign up for an AWS account

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

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

Sign in to the Amazon CloudWatch console

To sign in to the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, use the navigation bar to change the Region to the Region where you have your AWS resources.
3. Even if this is the first time you are using the CloudWatch console, **Your Metrics** could already report metrics, because you have used an AWS product that automatically pushes metrics to Amazon CloudWatch for free. Other services require that you enable metrics.

If you do not have any alarms, the **Your Alarms** section will have a **Create Alarm** button.

Set up the AWS CLI

You can use the AWS CLI or the Amazon CloudWatch CLI to perform CloudWatch commands. Note that the AWS CLI replaces the CloudWatch CLI; we include new CloudWatch features only in the AWS CLI.

For information about how to install and configure the AWS CLI, see [Getting Set Up with the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).

For information about how to install and configure the Amazon CloudWatch CLI, see [Set Up the Command Line Interface](#) in the [Amazon CloudWatch CLI Reference](#).

Getting started with Amazon CloudWatch

Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The CloudWatch overview home page appears.

CloudWatch: Overview ▾

All resources ▾

services summary ⓘ

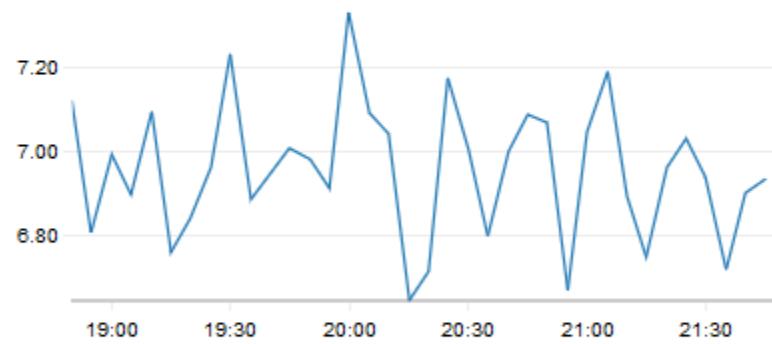
Services

Status	Alarm
🔴 EC2	1
🔴 Lambda	2
🔴 RDS	1
⚠️ Kinesis	-
✅ DynamoDB	-
❓ API Gateway	-
❓ Billing	-
❓ Classic ELB	-
❓ CloudFront	-
🕒 CloudWatch Events	-

Default dashboard ⓘ Edit dashboard

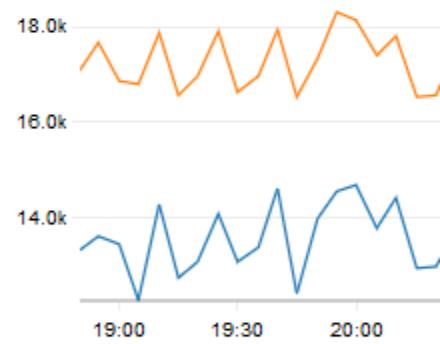
Custom metric 1

Percent



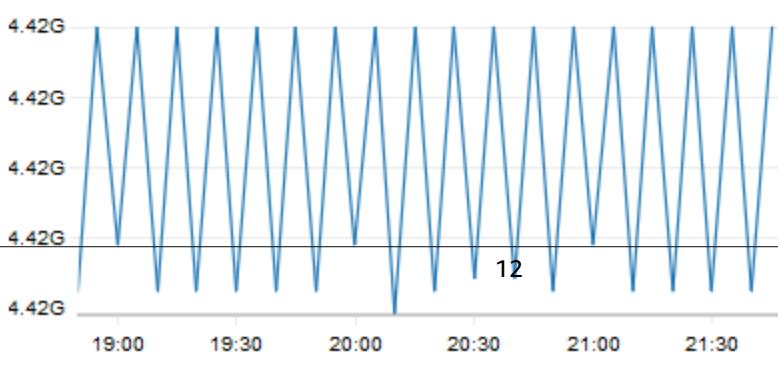
Custom metric 2

Bytes



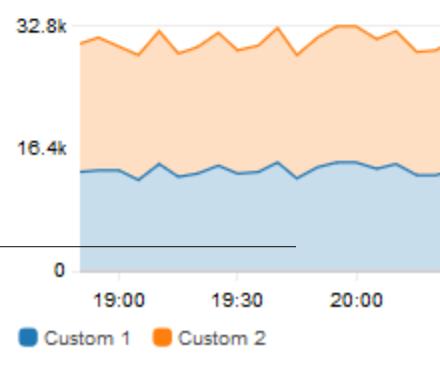
Custom metrics 5

Bytes



Custom metrics 2

Bytes



Custom 1 Custom 2

The overview displays the following items, refreshed automatically.

- The upper left shows a list of AWS services you use in your account, along with the state of alarms in those services. The upper right shows two or four alarms in your account, depending on how many AWS services you use. The alarms shown are those in the ALARM state or those that most recently changed state.

These upper areas enable you to assess the health of your AWS services, by seeing the alarm states in every service and the alarms that most recently changed state. This helps you monitor and quickly diagnose issues.

- Below these areas is the *default dashboard*, if one exists. The default dashboard is a custom dashboard that you have created and named **CloudWatch-Default**. This is a convenient way for you to add metrics about your own custom services or applications to the overview page, or to bring forward additional key metrics from AWS services that you most want to monitor.
- If you use six or more AWS services, below the default dashboard is a link to the automatic cross-service dashboard. The cross-service dashboard automatically displays key metrics from every AWS service you use, without requiring you to choose what metrics to monitor or create custom dashboards. You can also use it to drill down to any AWS service and see even more key metrics for that service.

If you use fewer than six AWS services, the cross-service dashboard is shown automatically on this page.

From this overview, you can focus your view to a specific resource group or a specific AWS service. This enables you to narrow your view to a subset of resources in which you are interested. Using resource groups enables you to use tags to organize projects, focus on a subset of your architecture, or just distinguish between your production and development environments. For more information, see [What Is AWS Resource Groups?](#).

Topics

- [See key metrics from all AWS services \(p. 13\)](#)
- [Focus on metrics and alarms in a single AWS service \(p. 15\)](#)
- [Focus on metrics and alarms in a resource group \(p. 16\)](#)

See key metrics from all AWS services

You can switch to the Cross-service dashboard screen and interact with dashboards for all of the AWS services that you're using. The CloudWatch Console displays your dashboards in alphabetical order and shows one or two key metrics on each dashboard.

Note

If you're using five or more AWS services, the CloudWatch Console won't display the Cross-service dashboard on the Overview screen.

To open the Cross-service dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
You're directed to the Overview screen.
2. From the Overview screen, select the dropdown that reads **Overview**, and then choose **Cross service dashboard**.

You're directed to the Cross service dashboard screen.

3. (Optional) If you're using the original interface, scroll to the section **Cross-service dashboard**, and then choose **View Cross-service dashboard**.

You're directed to the Cross-service dashboard screen.
4. You can focus on a particular service in two ways:
 - a. To see more key metrics for a service, choose its name from the list at the top of the screen, where **Cross service dashboard** is currently shown. Or, you can choose **View Service dashboard** next to the service name.

An automatic dashboard for that service is displayed, showing more metrics for that service. Additionally, for some services, the bottom of the service dashboard displays resources related to that service. You can choose one of those resources to that service console and focus further on that resource.
 - b. To see all the alarms related to a service, choose the button on the right of the screen next to that service name. The text on this button indicates how many alarms you have created in this service, and whether any are in the ALARM state.

When the alarms are displayed, multiple alarms that have similar settings (such as dimensions, threshold, or period) may be shown in a single graph.

You can then view details about an alarm and see the alarm history. To do so, hover on the alarm graph, and choose the actions icon, **View in alarms**.

The alarms view appears in a new browser tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

5. You can focus on resources in a particular resource group. To do so, choose the resource group from the list at the top of the page where **All resources** is displayed.

For more information, see [Focus on metrics and alarms in a resource group \(p. 16\)](#).
6. To change the time range shown in all graphs and alarms currently displayed, select the range you want next to **Time range** at the top of the screen. Choose **custom** to select from more time range options than those displayed by default.
7. Alarms are always refreshed once a minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose the refresh rate you want. You can also choose to turn off automatic refresh.

Remove a service from appearing in the cross-service dashboard

You can prevent a service's metrics from appearing in the cross-service dashboard. This helps you focus your cross-service dashboard on the services you most want to monitor.

If you remove a service from the cross-service dashboard, the alarms for that service still appear in the views of your alarms.

To remove a service's metrics from the cross-service dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The home page appears.
2. At the top of the page, under **Overview**, choose the service you want to remove.

The view changes to show metrics from only that service.

3. Choose **Actions**, then clear the check box next to **Show on cross service dashboard**.

Focus on metrics and alarms in a single AWS service

On the CloudWatch home page, you can focus the view to a single AWS service. You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure shows only how to focus on an AWS service.

To focus on a single service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The home page appears.

2. Choose the service name from the list at the top of the screen, where **Overview** is currently shown.

The view changes to display graphs of key metrics from the selected service.

3. To switch to viewing the alarms for this service, choose **Alarms dashboard** at the top of the screen where **Service dashboard** is currently displayed.
4. When viewing metrics, you can focus on a particular metric in several ways:

- a. To see more details about the metrics in any graph, hover on the graph, and choose the actions icon, **View in metrics**.

The graph appears in a new tab, with the relevant metrics listed below the graph. You can customize your view of this graph, changing the metrics and resources shown, the statistic, the period, and other factors to get a better understanding of the current situation.

- b. You can view log events from the time range shown in the graph. This may help you discover events that happened in your infrastructure that are causing an unexpected change in your metrics.

To see the log events, hover on the graph, and choose the actions icon, **View in logs**.

The CloudWatch Logs view appears in a new tab, displaying a list of your log groups. To see the log events in one of these log groups that occurred during the time range shown in the original graph, choose that log group.

5. When viewing alarms, you can focus on a particular alarm in several ways:

- To see more details about an alarm, hover on the alarm, and choose the actions icon, **View in alarms**.

The alarms view appears in a new tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

6. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
7. To change the time range shown in all graphs and alarms currently displayed, next to **Time range** at the top of the screen, choose the range. To select from more time range options than those displayed by default, choose **custom**.
8. To return to the cross-service dashboard, choose **Overview** in the list at the top of the screen that currently shows the service you are focusing on.

Alternatively, from any view, you can choose **CloudWatch** at the top of the screen to clear all filters and return to the overview page.

Focus on metrics and alarms in a resource group

You can focus your view to display metrics and alarms from a single resource group. Using resource groups enables you to use tags to organize projects, focus on a subset of your architecture, or distinguish between your production and development environments. They also enable you to focus on each of these resource groups on the CloudWatch overview. For more information, see [What Is AWS Resource Groups?](#).

When you focus on a resource group, the display changes to show only the services where you have tagged resources as part of this resource group. The recent alarms area shows only the alarms that are associated with resources that are part of the resource group. Additionally, if you have created a dashboard with the name **CloudWatch-Default-ResourceGroupName**, it is displayed in the **Default dashboard** area.

You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure shows just how to focus on a resource group.

To focus on a single resource group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. At the top of the page, where **All resources** is displayed, choose a resource group.
3. To see more metrics related to this resource group, near the bottom of the screen, choose **View cross service dashboard**.

The cross-service dashboard appears, showing only the services related to this resource group. For each service, one or two key metrics are displayed.

4. To change the time range shown in all graphs and alarms currently displayed, for **Time range** at the top of the screen, select a range. To select from more time range options than those displayed by default, choose **custom**.
5. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
6. To return to showing information about all the resources in your account, near the top of the screen where the name of the resource group is currently displayed, choose **All resources**.

Using CloudWatch with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples

For examples specific to CloudWatch, see [Code examples for CloudWatch using AWS SDKs \(p. 847\)](#).

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Using Amazon CloudWatch dashboards

Amazon CloudWatch dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those resources that are spread across different Regions. You can use CloudWatch dashboards to create customized views of the metrics and alarms for your AWS resources.

With dashboards, you can create the following:

- A single view for selected metrics and alarms to help you assess the health of your resources and applications across one or more Regions. You can select the color used for each metric on each graph, so that you can easily track the same metric across multiple graphs.

You can create dashboards that display graphs and other widgets from multiple AWS accounts and multiple Regions. For more information, see [Cross-account cross-Region CloudWatch console \(p. 284\)](#).

- An operational playbook that provides guidance for team members during operational events about how to respond to specific incidents.
- A common view of critical resource and application measurements that can be shared by team members for faster communication flow during operational events.

You can create dashboards by using the console, the AWS CLI, or the `PutDashboard` API.

To access CloudWatch dashboards, you need one of the following:

- The `AdministratorAccess` policy
- The `CloudWatchFullAccess` policy
- A custom policy that includes one or more of these specific permissions:
 - `cloudwatch:GetDashboard` and `cloudwatch>ListDashboards` to be able to view dashboards
 - `cloudwatch:PutDashboard` to be able to create or modify dashboards
 - `cloudwatch>DeleteDashboards` to be able to delete dashboards

Contents

- [Creating a CloudWatch dashboard \(p. 19\)](#)
- [Cross-account cross-Region dashboards \(p. 19\)](#)
- [Creating and working with widgets on CloudWatch dashboards \(p. 22\)](#)
- [Sharing CloudWatch dashboards \(p. 41\)](#)
- [Use live data \(p. 51\)](#)
- [Viewing an animated dashboard \(p. 51\)](#)
- [Add a dashboard to your Favorites list \(p. 52\)](#)
- [Change the period override setting or refresh interval for the CloudWatch dashboard \(p. 52\)](#)
- [Change the time range or time zone format of a CloudWatch dashboard \(p. 53\)](#)

Creating a CloudWatch dashboard

To get started with CloudWatch dashboards, you must first create a dashboard. You can create multiple dashboards. There is no limit on the number of CloudWatch dashboards in your AWS account. All dashboards are global, not Region-specific.

The steps in this section are for creating a dashboard using the console. You can also create a dashboard with the `PutDashboard` API, which uses a JSON string to define the dashboard contents. To create a dashboard using `PutDashboard` and base this dashboard on an existing dashboard, choose **Actions** and then **View/edit source** to display and copy the JSON string of a current dashboard to use for your new dashboard.

For more information about creating a dashboard using the API, see [PutDashboard](#) in the Amazon CloudWatch API Reference.

To create a dashboard using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and then **Create dashboard**.
3. In the **Create new dashboard** dialog box, enter a name for the dashboard and choose **Create dashboard**.

If you use the name **CloudWatch-Default**, the dashboard appears on the overview on the CloudWatch home page. For more information, see [Getting started with Amazon CloudWatch \(p. 11\)](#).

If you use resource groups and name the dashboard **CloudWatch-Default-ResourceGroupName**, it appears on the CloudWatch home page when you focus on that resource group.

4. Do one of the following in the **Add to this dashboard** dialog box:
 - To add a graph to your dashboard, choose **Line** or **Stacked area** and choose **Configure**. In the **Add metric graph** dialog box, select the metrics to graph and choose **Create widget**. If a specific metric doesn't appear in the dialog box because it hasn't published data in more than 14 days, you can add it manually. For more information, see [Graph metrics manually on a CloudWatch dashboard \(p. 25\)](#).
 - To add a number displaying a metric to the dashboard, choose **Number** and then **Configure**. In the **Add metric graph** dialog box, select the metrics to graph and choose **Create widget**.
 - To add a text block to your dashboard, choose **Text** and then **Configure**. In the **New text widget** dialog box, for **Markdown**, add and format your text using [Markdown](#). Choose **Create widget**.
5. Optionally, choose **Add widget** and repeat step 4 to add another widget to the dashboard. You can repeat this step multiple times.
6. Choose **Save dashboard**.

Cross-account cross-Region dashboards

You can create *cross-account cross-Region dashboards*, which summarize your CloudWatch data from multiple AWS accounts and multiple Regions into one dashboard. From this high-level dashboard you can get a view of your entire application, and also drill down into more specific dashboards without having to sign in and out of accounts or switch Regions.

You can create cross-account cross-Region dashboards in the AWS Management Console and programmatically.

Prerequisite

Before you can create a cross-account cross-Region dashboard, you must enable at least one sharing account and at least one monitoring account. Additionally, to be able to use the CloudWatch console to create a cross-account dashboard, you must enable the console for cross-account functionality. For more information, see [Cross-account cross-Region CloudWatch console \(p. 284\)](#).

Creating and using a cross-account cross-Region dashboard with the AWS Management Console

You can use the AWS Management Console to create a cross-account cross-Region dashboard.

To create a cross-account cross-Region dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Dashboards**.
4. Choose a dashboard, or create a new one.
5. At the top of the screen, you can switch between accounts and Regions. As you create your dashboard, you can include widgets from multiple accounts and Regions. Widgets include graphs, alarms, and CloudWatch Logs Insights widgets.

Creating a graph with metrics from different accounts and Regions

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Metrics**.
4. Under **All metrics**, choose the account and Region that you want to add metrics from.
5. Add the metrics you want to the graph. For more information, see [Graphing metrics \(p. 85\)](#).
6. Repeat steps 4-5 to add metrics from other accounts and Regions.
7. (Optional) Choose the **Graphed metrics** tab and add a metric math function that uses the metrics that you have chosen. For more information, see [Using metric math \(p. 97\)](#).

You can also set up a single graph to include multiple **SEARCH** functions. Each search can refer to a different account or Region.

8. When you are finished with the graph, choose **Actions, Add to dashboard**.

Select your cross-account dashboard, and choose **Add to dashboard**.

Adding an alarm from a different account to your cross-account dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. At the top of the page, choose the account where the alarm is located.
4. In the navigation pane, choose **Alarms**.
5. Select the check box next to the alarm that you want to add, and choose **Add to dashboard**.
6. Select the cross-account dashboard that you want to add it to, and choose **Add to dashboard**.

Create a cross-account cross-Region dashboard programmatically

You can use the AWS APIs and SDKs to create dashboards programmatically. For more information, see [PutDashboard](#).

To enable cross-account cross-Region dashboards, we have added new parameters to the dashboard body structure, as shown in the following table and examples. For more information about overall dashboard body structure, see [Dashboard Body Structure and Syntax](#).

Parameter	Use	Scope	Default
accountId	Specifies the ID of the account where the widget or the metric is located.	Widget or metric	Account that is currently logged in
region	Specifies the Region of the metric.	Widget or metric	Current Region selected in the console

The following examples illustrate the JSON source for widgets in a cross-account cross-Region dashboard.

This example sets the accountId field to the ID of the sharing account at the widget level. This specifies that all metrics in this widget will come from that sharing account and Region.

```
{  
  "widgets": [  
    {  
      ...  
      "properties": {  
        "metrics": [  
          ...  
          ],  
          "accountId": "111122223333",  
          "region": "us-east-1"  
        }  
      }  
    ]  
}
```

This example sets the accountId field differently at the level of each metric. In this example, the different metrics in this metric math expression come from different sharing accounts and different Regions.

```
{  
  "widgets": [  
    {  
      ...  
      "properties": {  
        "metrics": [  
          [ { "expression": "SUM(METRICS())", "label": "[avg: ${AVG}] Expression1", "id": "e1", "stat": "Sum" } ],  
          [ "AWS/EC2", "CPUUtilization", { "id": "m2", "accountId": "5555666677778888", "region": "us-east-1", "label": "[avg: ${AVG}] ApplicationALabel" } ],  
          [ ".", ".", { "id": "m1", "accountId": "9999000011112222", "region": "eu-west-1", "label": "[avg: ${AVG}] ApplicationBLabel" } ]  
        ]  
      }  
    }  
  ]  
}
```

```
        "view": "timeSeries",
        "stacked": false,
        "stat": "Sum",
        "period": 300,
        "title": "Cross account example"
    }
]
}
```

This example shows an alarm widget.

```
{
    "type": "metric",
    "x": 6,
    "y": 0,
    "width": 6,
    "height": 6,
    "properties": {
        "accountID": "111122223333",
        "title": "over50",
        "annotations": {
            "alarms": [
                "arn:aws:cloudwatch:us-east-1:379642911888:alarm:over50"
            ]
        },
        "view": "timeSeries",
        "stacked": false
    }
}
```

This example is for a CloudWatch Logs Insights widget.

```
{
    "type": "log",
    "x": 0,
    "y": 6,
    "width": 24,
    "height": 6,
    "properties": {
        "query": "SOURCE 'route53test' | fields @timestamp, @message\n| sort @timestamp
desc\n| limit 20",
        "accountId": "111122223333",
        "region": "us-east-1",
        "stacked": false,
        "view": "table"
    }
}
```

Another way to create dashboards programmatically is to first create one in the AWS Management Console, and then copy the JSON source of this dashboard. To do so, load the dashboard and choose **Actions, View/edit source**. You can then copy this dashboard JSON to use as a template to create similar dashboards.

Creating and working with widgets on CloudWatch dashboards

Use the topics in this section to create and work with graphs, alarms, and text widgets in your dashboards.

Contents

- [Add or remove a graph from a CloudWatch dashboard \(p. 23\)](#)
- [Graph metrics manually on a CloudWatch dashboard \(p. 25\)](#)
- [Working with existing graphs \(p. 25\)](#)
- [Add a metrics explorer widget to a CloudWatch dashboard \(p. 30\)](#)
- [Add an alarm widget to a CloudWatch dashboard \(p. 32\)](#)
- [Add a custom widget to a CloudWatch dashboard \(p. 32\)](#)
- [Add or remove a number widget from a CloudWatch dashboard \(p. 40\)](#)
- [Add or remove a text widget from a CloudWatch dashboard \(p. 40\)](#)
- [Link and unlink graphs on a CloudWatch dashboard \(p. 41\)](#)

Add or remove a graph from a CloudWatch dashboard

You can add graphs containing one or more metrics to your dashboard for the resources you monitor. You can remove the graphs when they're no longer needed.

To add a graph to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose the type of graph you want: **Line**, **Stacked area**, **Bar**, or **Pie**. Then choose **Next**.
5. Choose **Metrics** and then choose **Configure**.
6. In the **All metrics** tab, select the metrics to graph. If a specific metric doesn't appear in the dialog box because it hasn't published data in more than 14 days, you can add it manually. For more information, see [Graph metrics manually on a CloudWatch dashboard \(p. 25\)](#).
7. (Optional) To change the type of graph, choose **Graph options**. You can then choose between a line graph, stacked area chart, bar chart, pie chart, or number.
8. (Optional) As you choose metrics to graph, you can specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics** and then **Dynamic labels**.

By default, the dynamic values you add to the label appear at the beginning of the label. You can then choose the **Label** value for the metric to edit the label. For more information, see [Using dynamic labels \(p. 88\)](#).

9. (Optional) As you choose metrics to graph, you can change their color on the graph. To do so, choose **Graphed metrics** and select the color square next to the metric to display a color picker box. Choose another color square in the color picker. Click outside the color picker to see your new color on the graph. Alternatively, in the color picker, you can enter the six-digit standard HTML hex color code for the color you want and press **Enter**.
10. (Optional) To view more information about the metric being graphed, hover over the legend.
11. (Optional) To change the widget type, hover over the title area of the graph and choose **Widget actions**, **Widget type**.
12. (Optional) To change the statistic used for a metric, choose **Graphed metrics**, **Statistic**, and select the statistic you want to use. For more information, see [Statistics \(p. 5\)](#).
13. (Optional) To change the time range shown on the graph, choose either **custom** at the top of the graph or one of the time periods to the left of **custom**.

14. (Optional) Horizontal annotations help dashboard users quickly see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose **Graph options**, **Add horizontal annotation**:

- a. For **Label**, enter a label for the annotation.
- b. For **Value**, enter the metric value where the horizontal annotation appears.
- c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose **Above** or **Below** for the corresponding area to be filled. If you specify **Between**, another **Value** field appears, and the area of the graph between the two values is filled.
- d. For **Axis**, specify whether the numbers in **Value** refer to the metric associated with the left y-axis or the right y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

15. (Optional) Vertical annotations help you mark milestones in a graph, such as operational events or the beginning and end of a deployment. To add a vertical annotation, choose **Graph options**, **Add vertical annotation**:

- a. For **Label**, enter a label for the annotation. To show only the date and time on the annotation, keep the **Label** field blank.
- b. For **Date**, specify the date and time where the vertical annotation appears.
- c. For **Fill**, specify whether to use fill shading before or after a vertical annotation or between two vertical annotations. For example, choose **Before** or **After** for the corresponding area to be filled. If you specify **Between**, another **Date** field appears, and the area of the graph between the two values is filled.

Repeat these steps to add multiple vertical annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

16. Choose **Create widget**.

17. Choose **Save dashboard**.

To remove a graph from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Hover over the title of the graph and choose **Widget actions** and then **Delete**.
4. Choose **Save dashboard**. If you attempt to navigate away from the dashboard before you save your changes, you're prompted to either save or discard your changes.

Graph metrics manually on a CloudWatch dashboard

If a metric hasn't published data in the past 14 days, you can't find it when searching for metrics to add to a graph on a CloudWatch dashboard. Use the following steps to add any metric manually to an existing graph.

To add a metric that you can't find in search to a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. The dashboard must already contain a graph where you want to add the metric. If it doesn't, create the graph and add any metric to it. For more information, see [Add or remove a graph from a CloudWatch dashboard \(p. 23\)](#).
4. Choose **Actions, View/edit source**.

A JSON block appears. The block specifies the widgets on the dashboard and their contents. The following is an example of one part of this block, which defines one graph.

```
{  
    "type": "metric",  
    "x": 0,  
    "y": 0,  
    "width": 6,  
    "height": 3,  
    "properties": {  
        "view": "singleValue",  
        "metrics": [  
            [ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]  
        ],  
        "region": "us-west-1"  
    }  
},
```

In this example, the following section defines the metric shown on this graph.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]
```

5. Add a comma after the end bracket if there isn't already one and then add a similar bracketed section after the comma. In this new section, specify the namespace, metric name, and any necessary dimensions of the metric that you're adding to the graph. The following is an example.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ],  
[ "MyNamespace", "MyMetricName", "DimensionName", "DimensionValue" ]
```

For more information about the formatting of metrics in JSON, see [Properties of a Metric Widget Object](#).

6. Choose **Update**.

Working with existing graphs

Follow the procedures in these sections to edit and modify your existing dashboard graph widgets.

Topics

- [Edit a graph on a CloudWatch dashboard \(p. 26\)](#)
- [Move or resize a graph on a CloudWatch dashboard \(p. 30\)](#)

- [Rename a graph on a CloudWatch dashboard \(p. 30\)](#)

Edit a graph on a CloudWatch dashboard

You can change a graph's title, statistic, or period. You also can add and remove metrics from graphs. If your graph contains multiple metrics, you can hide metrics temporarily to reduce clutter.

New interface

To edit a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. In the upper right corner of the graph that you want to edit, choose **Widget actions**, and then choose **Edit**.
4. To change the title of the graph, choose the pen and paper icon next to the title. Enter the new title, and then choose **Apply**.
5. To change the time range that's shown on the graph, choose one of the predefined time ranges in the upper right corner. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
6. To change your graph's widget type, select the **Options** tab. You can choose **Line**, **Stacked area**, **Number**, **Bar**, or **Pie**.

Tip

You also can change your graph's widget type by choosing the dropdown that's next to the predefined time ranges.

7. In the **Graphed metrics** tab, you can change the dynamic label, label color, statistic, and period. You also can change the position of labels from left to right on the Y axis.

- a. (Optional) To specify a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display statistics about your metrics. By default, dynamic values appear at the beginning of your labels. After you specify a dynamic value for a label, you can select the label to edit label value. For more information about dynamic labels, see [Using dynamic labels \(p. 88\)](#).

Note

Dynamic labels update automatically whenever dashboards or graphs are refreshed.

- b. (Optional) To change the color of a metric, choose the colored square that's next to the metric. A box appears where you can choose a different color. Alternatively, you can enter a six-digit hex color code to specify the color that you want.
 - c. (Optional) To change the statistic, choose the statistic value under the **Statistic** column, and then choose a new value. For more information, see [Statistics \(p. 5\)](#).
 - d. (Optional) To change the period, choose the period value under the **Period** column, and then choose a new value.
8. To add or edit horizontal annotations, choose **Options**:
 - a. To add a horizontal annotation, choose **Add horizontal annotation**.
 - b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.
 - c. For **Value**, choose the pen and paper icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.
 - d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, another new label and value field appears.

Tip

You can change the fill color by choosing the colored square next to the annotation.

- e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.

Note

You can repeat these steps to add multiple horizontal annotations to the same graph.

- f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
- g. To delete an annotation, choose **X** under the **Actions** column.

9. To add or edit vertical annotations, choose **Options**:

- a. To add a vertical annotation, choose **Add vertical annotation**.
- b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

Tip

To show only the date and time, leave the label field blank.

- c. For **Date**, choose the current date and time. Then enter the new date and time.
- d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

Tip

You can change the fill color by choosing the color square next to the annotation.

Note

You can repeat these steps to add multiple vertical annotations to the same graph.

- e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
- f. To delete an annotation, choose **X** under the **Actions** column.
10. To customize the Y-axis, choose the **Options** tab. You can enter a custom label in the **Label** field under the **Left Y-axis** section. If the graph also displays values on the right Y-axis, you can customize that label, too. You can also set minimums and maximums on the Y-axis values so that the graph displays only the value range that you specify.
11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. In the upper right corner of the graph that you want to edit, select **Widget actions**. Choose **Legend**, and then choose **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. Select the colored square in the graph's footer. An **X** appears in the colored square when you hover over it, and it turns gray.
4. To restore the hidden metric, clear the **X** in the gray square.

Original interface

To edit a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. Hover over the upper right corner of the graph that you want to edit. Choose **Widget actions**, and then choose **Edit**.
4. To change the title of your graph, select the pencil icon next to the title. Enter the new title, and then press **Enter** on your keyboard.
5. To change the time range that's shown on the graph, choose one of the predefined time ranges in the upper right corner. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
6. To change your graph's widget type, select the **Graph options** tab. You can choose **Line**, **Stacked area**, **Number**, **Bar**, or **Pie**.

Tip

You also can change your graph's widget type by choosing the dropdown that's next to the predefined time ranges.

7. In the **Graphed metrics** tab, you can change the dynamic label, label colour, statistic, and period. You also can change the position of labels from left to right on the Y axis.
 - a. (Optional) To specify a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display statistics about your metrics. By default, dynamic values appear at the beginning of your labels. After you specify a dynamic value for a label, you can select the label to edit label value. For more information about dynamic labels, see [Using dynamic labels \(p. 88\)](#).

Note

Dynamic labels update automatically whenever dashboards or graphs are refreshed.

- b. (Optional) To change the color of a metric, choose the colored square that's next to the metric. A box appears where you can choose a different color. Alternatively, you can enter a six-digit hex color code to specify the color that you want.
- c. (Optional) To change the statistic, choose the statistic value under the **Statistic** column, and then choose a new value. For more information, see [Statistics \(p. 5\)](#).
- d. (Optional) To change the period, choose the period value under the **Period** column, and then choose a new value.

Note

Your new period setting is used only if the refresh interval rate on the dashboard is set to **Auto refresh**. Any other refresh interval rate setting overrides period settings for individual widgets.

8. To add or edit horizontal annotations, choose **Graph options**:
 - a. To add a horizontal annotation, choose **Add horizontal annotation**.
 - b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.
 - c. For **Value**, choose the pencil icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.
 - d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

Tip

You can change the fill color by choosing the color square next to the annotation.

- e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.

Note

You can repeat these steps to add multiple horizontal annotations to the same graph.

- f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
- g. To delete an annotation, choose **X** under the **Actions** column.

9. To add or edit vertical annotations, choose **Graph options**:

- a. To add a vertical annotation, choose **Add vertical annotation**.
- b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

Tip

To show only the date and time, leave the label field blank.

- c. For **Date**, choose the pencil icon next to the current date and time. Then enter the new date and time.
- d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

Tip

You can change the fill color by choosing the color square next to the annotation.

Note

You can repeat these steps to add multiple vertical annotations to the same graph.

- e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
- f. To delete an annotation, choose **X** under the **Actions** column.
10. To customize the Y-axis, choose **Graph options**. You can enter a custom label in **Label** under **Left Y-axis**. If the graph also displays values on the right Y-axis, you can customize that label, too. You can also set minimums and maximums on the Y-axis values, and the graph displays only the value range that you specify.
11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. Hover over the upper right corner of the graph that you want to edit, and then select **Widget actions**. Choose **Legend**, and then choose **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. Select the colored square in the graph's footer. An **X** appears in the colored square when you hover over it.
4. To restore the hidden metric, select the now greyed-out colored square.

Move or resize a graph on a CloudWatch dashboard

You can arrange and resize graphs on your CloudWatch dashboard.

To move a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Hover over the title of the graph until the selection icon appears. Select and drag the graph to a new location on the dashboard.
4. Choose **Save dashboard**.

To resize a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and choose a dashboard.
3. To increase or decrease the size, hover over the graph and drag the lower right corner of the graph. Stop dragging the corner when you have the size that you want.
4. Choose **Save dashboard**.

To enlarge a graph temporarily

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Select the graph. Alternatively, hover over the title of the graph and choose **Widget actions**, **Enlarge**.

Rename a graph on a CloudWatch dashboard

You can change the default name that CloudWatch assigns to a graph on your dashboard.

To rename a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Hover over the title of the graph and choose **Widget actions** and **Edit**.
4. On the **Edit graph** screen, near the top, choose the title of the graph.
5. For **Title**, enter a new name and choose **Ok** (check mark). In the lower-right corner of the **Edit graph** screen, choose **Update widget**.

Add a metrics explorer widget to a CloudWatch dashboard

Metrics explorer widgets include graphs of multiple resources that have the same tag, or share the same resource property such as an instance type. These widgets stay up to date, as resources that match are created or deleted. Adding metrics explorer widgets to your dashboard helps you to troubleshoot your environment more efficiently.

For example, you can monitor your fleet of EC2 instances by assigning tags that represent their environments, such as production or test. You can then use these tags to filter and aggregate the

operational metrics, such as `CPUUtilization`, to understand the health and performance of the EC2 instances that are associated with each tag.

The following steps explain how to add a metrics explorer widget to a dashboard using the console. You can also add it programmatically or by using AWS CloudFormation. For more information, see [Metrics Explorer Widget Object Definition](#) and [AWS::CloudWatch::Dashboard](#).

To add a metrics explorer widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the dashboard where you want to add the metrics explorer widget.
4. Choose **Add widget**.
5. Choose **Explorer** and then choose **Next**.

Note

You must be opted in to the new dashboard view to be able to add a Metrics Explorer widget. To opt in, choose **Dashboards** in the navigation pane, then choose **try out the new interface** in the banner at the top of the page.

6. Do one of the following:
 - To use a template, choose **Pre-filled Explorer widget** and then select a template to use.
 - To create a custom visualization, choose **Empty Explorer widget**.
7. Choose **Create**.

If you used a template, the widget appears on your dashboard with the selected metrics. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

If you did not use a template, continue to the following steps.

8. In the new widget under **Explorer**, in the **Metrics** box, choose a single metric or all the available metrics from a service.

After you choose a metric, you can optionally repeat this step to add more metrics.

9. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name and then choose the statistic that you want.
10. Under **From**, choose a tag or a resource property to filter your results.

After you do this, you can optionally repeat this step to choose more tags or resource properties.

If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

If you choose different properties or tags, such as the **Production** tag and the M5 instance type, only the resources that match all of these selections are displayed. This is treated as an AND operation.

11. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

Depending on how you choose to aggregate, the result may be a single time series or multiple time series.

12. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
13. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.

14. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

Add an alarm widget to a CloudWatch dashboard

To add an alarm widget to a dashboard, you have two options.

- You can add a single alarm in a widget, which displays both the graph of the alarm's metric and the alarm status.
- You can add an *alarm status widget*, which displays the status of multiple alarms in a grid. Only the alarm names and current status are displayed; the graphs are not displayed. Up to 100 alarms can be included in one alarm status widget.

To add a single alarm, including its graph, to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, select the alarm to add, and then choose **Add to Dashboard**.
3. Select a dashboard, choose a widget type (**Line**, **Stacked area**, or **Number**), and then choose **Add to dashboard**.
4. To see your alarm on the dashboard, choose **Dashboards** in the navigation pane and select the dashboard.
5. (Optional) To temporarily make an alarm graph larger, select the graph.
6. (Optional) To change the widget type, pause on the title of the graph, choose **Widget actions**, and then choose **Widget type**.

To add an alarm status widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose **Alarm status**.
5. Select the check boxes next to the alarms that you want to add to the widget, and then choose **Create widget**.
6. Choose **Add to dashboard**.

To remove an alarm widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Pause on the widget, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**. If you attempt to navigate away from the dashboard before you save your changes, you're prompted to either save or discard your changes.

Add a custom widget to a CloudWatch dashboard

A *custom widget* is a CloudWatch dashboard widget that can call any AWS Lambda function with custom parameters. It then displays the returned HTML or JSON. Custom widgets are a simple way to build a custom data view on a dashboard. If you can write Lambda code and create HTML, you can create a useful custom widget. Additionally, Amazon provides several pre-built custom widgets that you can create without any code.

When you create a Lambda function to use as a custom widget, we strongly recommend that you include the prefix **customWidget** in the function name. This helps you know which of your Lambda functions are safe to use when you add custom widgets to your dashboard.

Custom widgets behave like other widgets on your dashboard. They can be refreshed and auto-refreshed, resized, and moved around. They react to the time range of the dashboard.

If you have set up CloudWatch console cross-account functionality, you can add a custom widget created in one account to dashboards in other accounts. For more information, see [Cross-account cross-Region CloudWatch console \(p. 284\)](#).

You can also use custom widgets on your own website by using the CloudWatch dashboard sharing feature. For more information, see [Sharing CloudWatch dashboards \(p. 41\)](#).

Topics

- [Details about custom widgets \(p. 33\)](#)
- [Security and JavaScript \(p. 35\)](#)
- [Interactivity in the custom widget \(p. 35\)](#)
- [Create a custom widget \(p. 36\)](#)
- [Sample custom widgets \(p. 37\)](#)

Details about custom widgets

Custom widgets work as follows:

1. The CloudWatch dashboard calls the Lambda function containing the widget code. It passes in any custom parameters that are defined in the widget.
2. The Lambda function returns a string of HTML, JSON, or Markdown. Markdown is returned as JSON in the following format:

```
{ "markdown": "markdown content" }
```

3. The dashboard displays the returned HTML or JSON.

If the function returns HTML, most HTML tags are supported. You can use Cascading Style Sheets (CSS) styles and Scalable Vector Graphics (SVG) to build sophisticated views.

The default style of HTML elements such as links and tables follow the styling of CloudWatch dashboards. You can customize this style by using inline styles, using the `<style>` tag. You can also deactivate the default styles by including a single HTML element with the class of `cwdb-no-default-styles`. The following example deactivates default styles: `<div class="cwdb-no-default-styles"></div>`.

Every call by a custom widget to Lambda includes a `widgetContext` element with the following contents, to provide the Lambda function developer with useful context information.

```
{  
    "widgetContext": {  
        "dashboardName": "Name-of-current-dashboard",  
        "widgetId": "widget-16",  
        "accountId": "012345678901",  
        "locale": "en",  
        "timezone": {  
            "label": "UTC",  
            "offsetISO": "+00:00",  
            "offsetInMinutes": 0  
        }  
    }  
}
```

```
        },
        "period": 300,
        "isAutoPeriod": true,
        "timeRange": {
            "mode": "relative",
            "start": 1627236199729,
            "end": 1627322599729,
            "relativeStart": 86400012,
            "zoom": {
                "start": 1627276030434,
                "end": 1627282956521
            }
        },
        "theme": "light",
        "linkCharts": true,
        "title": "Tweets for Amazon website problem",
        "forms": {
            "all": {}
        },
        "params": {
            "original": "param-to-widget"
        },
        "width": 588,
        "height": 369
    }
}
```

Default CSS styling

Custom widgets provide the following default CSS styling elements:

- You can use the CSS class **btn** to add a button. It turns an anchor () into a button as in the following example:

```
<a class="btn" href="https://amazon.com">Open Amazon</a>
```

- You can use the CSS class **btn btn-primary** to add a primary button.
- The following elements are styled by default: **table**, **select**, **headers (h1, h2, and h3)**, **preformatted text (pre)**, **input**, and **text area**.

Using the `describe` parameter

We strongly recommend that you support the **describe** parameter in your functions, even if it just returns an empty string. If you don't support it, and it is called in the custom widget, it displays widget content as if it was documentation.

If you include the **describe** parameter, the Lambda function returns the documentation in Markdown format and does nothing else.

When you create a custom widget in the console, after you select the Lambda function a **Get documentation** button appears. If you choose this button, the function is invoked with the **describe** parameter and the function's documentation is returned. If the documentation is well-formatted in markdown, CloudWatch parses the first entry in the documentation that is surrounded by three single backtick characters (` `` `) in YAML. Then, it automatically populates the documentation in the parameters. The following is an example of such well-formatted documentation.

```
``` yaml
echo: <h1>Hello world</h1>
````
```

Security and JavaScript

For security reasons, JavaScript is not allowed in the returned HTML. Removing the JavaScript prevents permission escalation issues, where the writer of the Lambda function injects code that could run with higher permissions than the user viewing the widget on the dashboard.

If the returned HTML contains any JavaScript code or other known security vulnerabilities, it is cleaned from the HTML before it is rendered on the dashboard. For example, the `<iframe>` and `<use>` tags are not allowed and are removed.

Custom Widgets won't run by default in a dashboard. Instead, you must explicitly allow a custom widget to run if you trust the Lambda function that it invokes. You can choose to allow it once or allow always, for both individual widgets and entire dashboard. You can also deny permission for individual widgets and the entire dashboard.

Interactivity in the custom widget

Even though JavaScript is not allowed, there are other ways to allow interactivity with the returned HTML.

- Any element in the returned HTML can be tagged with special configuration in a `<cwdb-action>` tag, which can display information in pop-ups, ask for confirmation on clicks, and call any Lambda function when that element is chosen. For example, you can define buttons that call any AWS API using a Lambda function. The returned HTML can be set to either replace the existing Lambda widget's content, or display inside a modal.
- The returned HTML can include links that open new consoles, open other customer pages, or load other dashboards.
- The HTML can include the `title` attribute for an element, which gives additional information if the user pauses on that element.
- The element can include CSS selectors, such as `:hover`, which can invoke animations or other CSS effects. You can also show or hide elements in the page.

`<cwdb-action>` definition and usage

The `<cwdb-action>` element defines a behavior on the immediately previous element. The content of the `<cwdb-action>` is either HTML to display or a JSON block of parameters to pass to a Lambda function.

The following is an example of a `<cwdb-action>` element.

```
<cwdb-action
    action="call|html"
    confirmation="message"
    display="popup|widget"
    endpoint="<lambda ARN>"
    event="click|dblclick|mouseenter">
    html | params in JSON
</cwdb-action>
```

- **action**— Valid values are `call`, which calls a Lambda function, and `html`, which displays any HTML contained within `<cwdb-action>`. The default is `html`.
- **confirmation**— Displays a confirmation message that must be acknowledged before the action is taken, allowing the customer to cancel.
- **display**— Valid values are `popup` and `widget`, which replaces the content of the widget itself. The default is `widget`.

- **endpoint**— The Amazon Resource Name (ARN) of the Lambda function to call. This is required if `action` is `call`.
- **event**— Defines the event on the previous element that invokes the action. Valid values are `click`, `dblclick`, and `mouseenter`. The `mouseenter` event can be used only in combination with the `html` action. The default is `click`.

Examples

The following is an example of how to use the `<cwdb-action>` tag to create a button that reboots an Amazon EC2 instance using a Lambda function call. It displays the success or failure of the call in a pop-up.

```
<a class="btn">Reboot Instance</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:rebootInstance" display="popup">
    { "instanceId": "i-342389adbfef" }
</cwdb-action>
```

The next example displays more information in a pop-up.

```
<a>Click me for more info in popup</a>
<cwdb-action display="popup">
    <h1>Big title</h1>
    More info about <b>something important</b>.
</cwdb-action>
```

This example is a **Next** button that replaces the content of a widget with a call to a Lambda function.

```
<a class="btn btn-primary">Next</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-east-1:123456:function:nextPage">
    { "pageNum": 2 }
</cwdb-action>
```

Create a custom widget

To create a custom widget, you can use one of the samples provided by Amazon, or you can create your own. The Amazon samples include samples in both JavaScript and Python, and are created by a AWS CloudFormation stack. For a list of samples, see [Sample custom widgets \(p. 37\)](#).

To create a custom widget in a CloudWatch dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. Choose **Add widget**.
4. Choose **Custom widget**.
5. Use one of the following methods:
 - To use a sample custom widget provided by Amazon, do the following:
 - a. Select the sample in the dropdown box.

The AWS CloudFormation console launches in a new browser. In the AWS CloudFormation console, do the following:

- b. (Optional) Customize the AWS CloudFormation stack name.
- c. Make selections for any parameters used by the sample.

- d. Select **I acknowledge that AWS CloudFormation might create IAM resources**, and choose **Create stack**.
- To create your own custom widget provided by Amazon, do the following:
 - a. Choose **Next**.
 - b. Choose to either select your Lambda function from a list, or enter its Amazon Resource Name (ARN). If you select it from a list, also specify the Region where the function is and the version to use.
 - c. For **Parameters**, make selections for any parameters used by the function.
 - d. Enter a title for the widget.
 - e. For **Update on**, configure when the widget should be updated (when the Lambda function should be called again). This can be one or more of the following: **Refresh** to update it when the dashboard auto-refreshes, **Resize** to update it whenever the widget is resized, or **Time Range** to update it whenever the dashboard's time range is adjusted, including when graphs are zoomed into.
 - f. If you are satisfied with the preview, choose **Create widget**.

Sample custom widgets

AWS provides sample custom widgets in both JavaScript and Python. You can create these sample widgets by using the link for each widget in this list. Alternatively, you can create and customize a widget by using the CloudWatch console. The links in this list open an AWS CloudFormation console and use an AWS CloudFormation quick-create link to create the custom widget.

You can also access the custom widget samples on [GitHub](#).

Following this list, complete examples of the Echo widget are shown for each language.

JavaScript

Sample custom widgets in JavaScript

- [Echo](#) – A basic echoer that you can use to test how HTML appears in a custom widget, without having to write a new widget.
- [Hello world](#) – A very basic starter widget.
- [Custom widget debugger](#) – A debugger widget that displays useful information about the Lambda runtime environment.
- [Query CloudWatch Logs Insights](#) – Run and edit CloudWatch Logs Insights queries.
- [Run Amazon Athena queries](#) – Run and edit Athena queries.
- [Call AWS API](#) – Call any read-only AWS API and display the results in JSON format.
- [Fast CloudWatch bitmap graph](#) – Render CloudWatch graphs using on the server side, for fast display.
- [Text widget from CloudWatch dashboard](#) – Displays the first text widget from the specified CloudWatch dashboard.
- [CloudWatch metric data as a table](#) – Displays raw CloudWatch metric data in a table.
- [Amazon EC2 table](#) – Displays the top EC2 instances by CPU utilization. This widget also includes a Reboot button, which is disabled by default.
- [AWS CodeDeploy deployments](#) – Displays CodeDeploy deployments.
- [AWS Cost Explorer report](#) – Displays a report on the cost of each AWS service for a selected time range.
- [Display content of external URL](#) – Displays the content of an externally accessible URL.

- [Display an Amazon S3 object](#) – Displays an object in an Amazon S3 bucket in your account.
- [Simple SVG pie chart](#) – Example of a graphical SVG-based widget.

Python

Sample custom widgets in Python

- [Echo](#) – A basic echoer which you can use to test how HTML appears in a custom widget, without having to write a new widget.
- [Hello world](#) – A very basic starter widget.
- [Custom widget debugger](#) – A debugger widget that displays useful information about the Lambda runtime environment.
- [Call AWS API](#) – Call any read-only AWS API and display the results in JSON format.
- [Fast CloudWatch bitmap graph](#) – Render CloudWatch graphs using on the server side, for fast display.
- [Send dashboard snapshot by email](#) – Take a snapshot of the current dashboard and send it to email recipients.
- [Send dashboard snapshot to Amazon S3](#) – Take a snapshot of the current dashboard and store it in Amazon S3.
- [Text widget from CloudWatch dashboard](#) – Displays the first text widget from the specified CloudWatch dashboard.
- [Display content of external URL](#) – Displays the content of an externally accessible URL.
- [RSS reader](#) – Displays RSS feeds.
- [Display an Amazon S3 object](#) – Displays an object in an Amazon S3 bucket in your account.
- [Simple SVG pie chart](#) – Example of a graphical SVG-based widget.

Echo widget in JavaScript

The following is the Echo sample widget in JavaScript.

```
const DOCS = `## Echo
A basic echo script. Anything passed in the ``echo`` parameter is returned as the
content of the custom widget.
### Widget parameters
Param	Description
**echo** | The content to echo back

### Example parameters
```
echo: <h1>Hello world</h1>
```;
;

exports.handler = async (event) => {
    if (event.describe) {
        return DOCS;
    }

    let widgetContext = JSON.stringify(event.widgetContext, null, 4);
    widgetContext = widgetContext.replace(/</g, '&lt;');
    widgetContext = widgetContext.replace(/>/g, '&gt;');

    return `${event.echo || ''}<pre>${widgetContext}</pre>`;
```

```
};
```

Echo widget in Python

The following is the Echo sample widget in Python.

```
import json

DOCS = """
## Echo
A basic echo script. Anything passed in the `echo` parameter is returned as the content
of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
``` yaml
echo: <h1>Hello world</h1>
```
"""

def lambda_handler(event, context):
    if 'describe' in event:
        return DOCS

    echo = event.get('echo', '')
    widgetContext = event.get('widgetContext')
    widgetContext = json.dumps(widgetContext, indent=4)
    widgetContext = widgetContext.replace('<', '&lt;')
    widgetContext = widgetContext.replace('>', '&gt;')

    return f'{echo}<pre>{widgetContext}</pre>'
```

Echo widget in Java

The following is the Echo sample widget in Java.

```
package example;

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

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Handler implements RequestHandler<Event, String>{

    static String DOCS = "
        + "## Echo\n"
        + "A basic echo script. Anything passed in the `echo` parameter is returned as the
content of the custom widget.\n"
        + "### Widget parameters\n"
        + "Param | Description\n"
        + "---|---\n"
        + "**echo** | The content to echo back\n\n"
        + "### Example parameters\n"
        + "```yaml\n"
        + "echo: <h1>Hello world</h1>\n"
        + "```\n";

    Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

```
@Override
public String handleRequest(Event event, Context context) {

    if (event.describe) {
        return DOCS;
    }

    return (event.echo != null ? event.echo : "") + "<pre>" +
gson.toJson(event.widgetContext) + "</pre>";
}
}

class Event {

    public boolean describe;
    public String echo;
    public Object widgetContext;

    public Event() {}

    public Event(String echo, boolean describe, Object widgetContext) {
        this.describe = describe;
        this.echo = echo;
        this.widgetContext = widgetContext;
    }
}
```

Add or remove a number widget from a CloudWatch dashboard

A number widget displays the current value of a selected metric, in numerals.

To add a number widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose **Number** and then **Configure**.
5. In the **Add metric graph** dialog box, select the metrics to graph and choose **Create widget**.
6. Choose **Save dashboard**.

To remove a number widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Pause over the upper-right corner of the number widget and choose **Widget actions** and then **Delete**.
4. Choose **Save dashboard**.

Add or remove a text widget from a CloudWatch dashboard

A text widget contains a block of text in **Markdown** format. You can add, edit, or remove text widgets from your CloudWatch dashboard.

To add a text widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose **Text** and then **Configure**.
5. For **Markdown**, add and format your text using **Markdown** and choose **Create widget**.
6. Choose **Save dashboard**.

To edit a text widget on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions** and then **Edit**.
4. Update the text as needed and choose **Update widget**.
5. Choose **Save dashboard**.

To remove a text widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions** and then **Delete**.
4. Choose **Save dashboard**.

Link and unlink graphs on a CloudWatch dashboard

You can link the graphs on your dashboard together, so that when you zoom in or zoom out on one graph, the other graphs zoom in or zoom out at the same time. You can unlink graphs to limit zoom to one graph.

To link the graphs on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Actions** and then **Link graphs**.

To unlink the graphs on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Clear **Actions** and then **Link graphs**.

Sharing CloudWatch dashboards

You can share your CloudWatch dashboards with people who do not have direct access to your AWS account. This enables you to share dashboards across teams, with stakeholders, and with people external

to your organization. You can even display dashboards on big screens in team areas, or embed them in Wikis and other webpages.

Warning

All people who you share the dashboard with are granted the permissions listed in [Permissions that are granted to people who you share the dashboard with \(p. 43\)](#) for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.

The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share dashboards, you can designate who can view the dashboard in three ways:

- Share a single dashboard and designate specific email addresses of the people who can view the dashboard. Each of these users creates their own password that they must enter to view the dashboard.
- Share a single dashboard publicly, so that anyone who has the link can view the dashboard.
- Share all the CloudWatch dashboards in your account and specify a third-party single sign-on (SSO) provider for dashboard access. All users who are members of this SSO provider's list can access all the dashboards in the account. To enable this, you integrate the SSO provider with Amazon Cognito. The SSO provider must support Security Assertion Markup Language (SAML). For more information about Amazon Cognito, see [What is Amazon Cognito?](#)

Permissions required to share a dashboard

To be able to share dashboards using any of the following methods and to see which dashboards have already been shared, you must be logged on to an IAM user or IAM role that has certain permissions.

To be able to share dashboards, your IAM user or IAM role must include the permissions included in the following policy statement:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "iam:CreateRole",  
        "iam:CreatePolicy",  
        "iam:AttachRolePolicy",  
        "iam:PassRole"  
    ],  
    "Resource": [  
        "arn:aws:iam::*:role/service-role/CWDBSharing*",  
        "arn:aws:iam::*:policy/*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "cognito-idp:*",  
        "cognito-identity:*"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "cloudwatch:GetDashboard",  
    ],  
}
```

```
"Resource": [  
    "*"  
    // or the ARNs of dashboards that you want to share  
]  
}
```

To be able to see which dashboards are shared, but not be able to share dashboards, your IAM user or IAM role can include a policy statement similar to the following:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "cognito-identity:*",  
        "cognito-idp:*"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "cloudwatch:ListDashboards",  
    ],  
    "Resource": [  
        "*"  
    ]  
}
```

Permissions that are granted to people who you share the dashboard with

When you share a dashboard, CloudWatch creates an IAM role in the account which gives the following permissions to the people who you share the dashboard with:

- `cloudwatch:GetInsightRuleReport`
- `cloudwatch:GetMetricData`
- `cloudwatch:DescribeAlarms`
- `ec2:DescribeTags`

Warning

All people who you share the dashboard with are granted these permissions for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.

The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share a dashboard, by default the permissions that CloudWatch creates restrict access to only the alarms and Contributor Insights rules that are on the dashboard when it is shared. If you add new alarms or Contributor Insights rules to the dashboard and want them to also be seen by the people who you shared the dashboard with, you must update the policy to allow these resources.

Share a single dashboard with specific users

Use the steps in this section to share a dashboard with specific email addresses that you choose.

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see logs table widgets \(p. 49\)](#).

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see composite alarms \(p. 48\)](#).

To share a dashboard with specific users

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions, Share dashboard**.
5. Next to **Share your dashboard and require a username and password**, choose **Start sharing**.
6. Under **Add email addresses**, enter the email addresses that you want to share the dashboard with.
7. When you have all the email addresses entered, read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then paste this link into email and send it to the invited users. They automatically receive a separate email with their user name and a temporary password to use to connect to the dashboard.

Share a single dashboard publicly

Follow the steps in this section to share a dashboard publicly. This can be useful to display the dashboard on a big screen in a team room, or embed it in a Wiki page.

Important

Sharing a dashboard publicly makes it accessible to anyone who has the link, with no authentication. Do this only for dashboards that do not contain sensitive information.

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see logs table widgets \(p. 49\)](#).

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see composite alarms \(p. 48\)](#).

To share a dashboard publicly

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions, Share dashboard**.
5. Next to **Share your dashboard publicly**, choose **Start sharing**.
6. Enter **Confirm** in the text box.
7. Read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.

9. On the next page, choose **Copy link to clipboard**. You can then share this link. Anyone you share the link with can access the dashboard, without providing credentials.

Share all CloudWatch dashboards in the account by using SSO

Use the steps in this section to share all the dashboards in your account with users by using single sign-on (SSO).

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see logs table widgets \(p. 49\)](#).

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see composite alarms \(p. 48\)](#).

To share your CloudWatch dashboards with users who are in an SSO provider's list

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions, Share dashboard**.
5. Choose **Go to CloudWatch Settings**.
6. If the SSO provider that you want isn't listed in **Available SSO providers**, choose **Manage SSO providers** and follow the instructions in [Set up SSO for CloudWatch dashboard sharing \(p. 45\)](#).

Then return to the CloudWatch console and refresh the browser. The SSO provider that you enabled should now appear in the list.

7. Choose the SSO provider that you want in the **Available SSO providers** list.
8. Choose **Save changes**.

Set up SSO for CloudWatch dashboard sharing

To set up dashboard sharing through a third-party single sign-on provider that supports SAML, follow these steps.

Important

We strongly recommend that you do not share dashboards using a non-SAML SSO provider. Doing so causes a risk of inadvertently allowing third parties to access your account's dashboards.

To set up an SSO provider to enable dashboard sharing

1. Integrate the SSO provider with Amazon Cognito. For more information, see [Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools](#).
2. Download the metadata XML file from your SSO provider.
3. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
4. In the navigation pane, choose **Settings**.
5. In the **Dashboard sharing** section, choose **Configure**.
6. Choose **Manage SSO providers**.

This opens the Amazon Cognito console in the US East (N. Virginia) Region (us-east-1). If you don't see any **User Pools**, the Amazon Cognito console might have opened in a different Region. If so, change the Region to **US East (N. Virginia) us-east-1** and proceed with the next steps.

7. Choose the **CloudWatchDashboardSharing** pool.
8. In the navigation pane, choose **Identity providers**.
9. Choose **SAML**.
10. Enter a name for your SSO provider in **Provider name**.
11. Choose **Select file**, and select the metadata XML file that you downloaded in step 1.
12. Choose **Create provider**.

See how many of your dashboards are shared

You can use the CloudWatch console to see how many of your CloudWatch dashboards are currently being shared with others.

To see how many of your dashboards are being shared

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. The **Dashboard sharing** section displays how many dashboards are shared.
4. To see which dashboards are shared, choose **number dashboards shared** under **Username and password** and under **Public dashboards**.

See which of your dashboards are shared

You can use the CloudWatch console to see which of your dashboards are currently being shared with others.

To see which of your dashboards are being shared

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. In the list of dashboards, see the **Share** column. Dashboards that have the icon in this column filled in are currently being shared.
4. To see which users a dashboard is being shared with, choose the dashboard name, and then choose **Actions, Share dashboard**.

The **Share dashboard *dashboard name*** page displays how the dashboard is being shared. If you want, you can stop sharing the dashboard by choosing **Stop sharing**.

Stop sharing one or more dashboards

You can stop sharing a single shared dashboard, or stop sharing all shared dashboards at once.

To stop sharing a single dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.

3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Choose **Stop sharing**.
6. In the confirmation box, choose **Stop sharing**.

To stop sharing all shared dashboards

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. In the **Dashboard sharing** section, choose **Stop sharing all dashboards**.
4. In the confirmation box, choose **Stop sharing all dashboards**.

Review shared dashboard permissions and change permission scope

Use the steps in this section if you want to review the permissions of the users of your shared dashboards, or change the scope of shared dashboard permissions.

To review shared dashboard permissions

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. (Optional) To limit which alarms that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:DescribeAlarms` permission from its current position to a new `Allow` statement that lists the ARNs of only the alarms that you want to be seen by shared dashboard users. See the following example.

```
{  
    "Effect": "Allow",  
    "Action": "cloudwatch:DescribeAlarms",  
    "Resource": [  
        "AlarmARN1",  
        "AlarmARN2"  
    ]  
}
```

If you do this, be sure to remove the `cloudwatch:DescribeAlarms` permission from a section of the current policy that looks like this:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "cloudwatch:GetInsightRuleReport",  
        "cloudwatch:GetMetricData",  
        "cloudwatch:DescribeAlarms",  
        "ec2:DescribeTags"  
    ],  
}
```

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

8. (Optional) To limit the scope of what Contributor Insights rules that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:GetInsightRuleReport` from its current position to a new Allow statement that lists the ARNs of only the Contributor Insights rules that you want to be seen by shared dashboard users. See the following example.

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:GetInsightRuleReport",
    "Resource": [
        "PublicContributorInsightsRuleARN1",
        "PublicContributorInsightsRuleARN2"
    ]
}
```

If you do this, be sure to remove `cloudwatch:GetInsightRuleReport` from a section of the current policy that looks like this:

```
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:GetInsightRuleReport",
        "cloudwatch:GetMetricData",
        "cloudwatch:DescribeAlarms",
        "ec2:DescribeTags"
    ],
    "Resource": "*"
}
```

Allowing people that you share with to see composite alarms

When you share a dashboard, by default the composite alarm widgets on the dashboard are not visible to the people who you share the dashboard with. For composite alarm widgets to be visible, you need to add a `DescribeAlarms: *` permission to the dashboard sharing policy. That permission would look like this:

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
}
```

Warning

The preceding policy statement give access to all alarms in the account. To reduce the scope of `cloudwatch:DescribeAlarms`, you must use a Deny statement. You can add a Deny statement to the policy and specify the ARNs of the alarms that you want to lock down. That deny statement should look similar to the following:

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
},
```

```
{  
    "Effect": "Deny",  
    "Action": "cloudwatch:DescribeAlarms",  
    "Resource": [  
        "SensitiveAlarm1ARN",  
        "SensitiveAlarm1ARN"  
    ]  
}
```

Allowing people that you share with to see logs table widgets

When you share a dashboard, by default the CloudWatch Logs Insights widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both CloudWatch Logs Insights widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see CloudWatch Logs widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the CloudWatch Logs widgets

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the log groups that you want shared. See the following example.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "logs:FilterLogEvents",  
        "logs:StartQuery",  
        "logs:StopQuery",  
        "logs:GetLogRecord"  
    ],  
    "Resource": [  
        "SharedLogGroup1ARN",  
        "SharedLogGroup2ARN"  
    ]  
},
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes those four permissions with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the log groups that you want shared. For example, if your Resource section for these permissions was the following:

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

Change the policy to specify only the ARNs of the log groups that you want shared, as in the following example:

```
"Resource": [  
    "SharedLogGroup1ARN",  
    "SharedLogGroup2ARN"  
]
```

Allowing people that you share with to see custom widgets

When you share a dashboard, by default the custom widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both custom widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see custom widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the custom widgets

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the Lambda functions that you want shared. See the following example.

```
{  
    "Sid": "Invoke",  
    "Effect": "Allow",  
    "Action": [  
        "lambda:InvokeFunction"  
    ],  
    "Resource": [  
        "LambdaFunction1ARN",  
        "LambdaFunction2ARN"  
    ]  
}
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes that permission with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the Lambda functions that you want shared. For example, if your Resource section for these permissions was the following:

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

Change the policy to specify only the ARNs of the custom widgets that you want shared, as in the following example:

```
"Resource": [  
    "LambdaFunction1ARN",  
    "LambdaFunction2ARN"  
]
```

Use live data

You can choose whether your metric widgets display *live data*. Live data is data published within the last minute that has not been fully aggregated.

- If live data is turned **off**, only data points with an aggregation period of at least one minute in the past are shown. For example, when using 5-minute periods, the data point for 12:35 would be aggregated from 12:35 to 12:40, and displayed at 12:41.
- If live data is turned **on**, the most recent data point is shown as soon as any data is published in the corresponding aggregation interval. Each time you refresh the display, the most recent data point may change as new data within that aggregation period is published. If you use a cumulative statistic such as **Sum** or **Sample Count**, using live data may result in a dip at the end of your graph.

You can choose to enable live data for a whole dashboard, or for individual widgets on the dashboard.

To choose whether to use live data on your entire dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. To permanently turn live data on or off for all widgets on the dashboard, do the following:
 - a. Choose **Actions, Settings, Bulk update live data**.
 - b. Choose **Live Data on** or **Live Data off**, and choose **Set**.
4. To temporarily override the live data settings of each widget, choose **Actions**. Then, under **Overrides**, next to **Live data**, do one of the following:
 - Choose **On** to temporarily turn on live data for all widgets.
 - Choose **Off** to temporarily turn off live data for all widgets.
 - Choose **Do not override** to preserve each widget's live data setting.

To choose whether to use live data on a single widget

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Select a widget, and choose **Actions, Edit**.
4. Choose the **Graph options** tab.
5. Select or clear the check box under **Live Data**.

Viewing an animated dashboard

You can view an animated dashboard that replays CloudWatch metric data that was captured over time. This can help you see trends, make presentations, or analyze issues after they occur.

Animated widgets in the dashboard include line widgets, stacked area widgets, number widgets, and metrics explorer widgets. Pie graphs, bar charts, text widgets, and logs widgets are displayed in the dashboard but are not animated.

To view an animated dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.

3. Choose the name of the dashboard.
4. Choose **Actions**, **Replay dashboard**.
5. (Optional) By default, when you start the animation, it appears as a sliding window. If you want the animation to appear as a point-by-point animation instead, choose the magnifying glass icon while the animation is paused and reset the zoom.
6. To start the animation, choose the Play button. You can also choose the back and forward buttons to move to other points in time.
7. (Optional) To change the time window for the animation, choose the calendar and select the time period.
8. To change the speed of the animation, choose **Auto speed** and select the new speed.
9. When you are finished, choose **Exit animate**.

Add a dashboard to your Favorites list

You can add a CloudWatch dashboard to a list of favorite dashboards to help you find it quickly. The **Favorites** list appears at the bottom of the navigation pane.

To add a dashboard to the Favorites list

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Select the star symbol next to the dashboard to add it.

Change the period override setting or refresh interval for the CloudWatch dashboard

You can specify how the period setting of graphs added to this dashboard are retained or modified.

To change the period override options

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Actions**.
3. Under **Period override**, choose one of the following:
 - Choose **Auto** to have the period of the metrics on each graph automatically adapt to the dashboard's time range.
 - Choose **Do not override** to ensure that the period setting of each graph is always obeyed.
 - Choose one of the other options to cause graphs added to the dashboard to always adapt that chosen time as their period setting.

The **Period override** always reverts to **Auto** when the dashboard is closed or the browser is refreshed. Different settings for **Period override** can't be saved.

You can change how often the data on your CloudWatch dashboard is refreshed or set it to automatically refresh.

To change the dashboard refresh interval

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. On the **Refresh options** menu (upper-right corner), choose **10 Seconds, 1 Minute, 2 Minutes, 5 Minutes, or 15 Minutes**.

To automatically refresh the dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Refresh options** and then **Auto refresh**.

Change the time range or time zone format of a CloudWatch dashboard

You can change the time range to display dashboard data over minutes, hours, days, or weeks. You also can change the time zone format to display dashboard data in UTC or local time. Local time is the time zone that's specified in your computer's operating system.

Note

If you create a dashboard with graphs that contain 100 or more high-resolution metrics, we recommend that you don't set the time range to longer than 1 hour. For more information, see [High-resolution metrics \(p. 95\)](#).

New console

To change the dashboard time range

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. From the dashboard screen, do one of the following:
 - In the upper right corner, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h, 3h, 12h, 1d, or 1w**).
 - Alternatively, you can choose one of the following custom time range options:
 - Choose **Custom**, and then choose the **Relative** tab. Choose a time range from 1 minute to 15 months.
 - Choose **Custom**, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

Tip

If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. In the upper right corner of the dashboard screen, choose **Custom**.
4. In the upper right corner of the box that appears, select **UTC** or **Local time** from the dropdown.
5. Choose **Apply**.

Old console

To change the dashboard time range

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. From the dashboard screen, do one of the following:
 - In the upper right corner, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).
 - Alternatively, you can choose one of the following custom time range options:
 - Choose the **custom** dropdown, and then choose the **Relative** tab. Select one of the predefined ranges, which span from 1 minute to 15 months.
 - Choose the **custom** dropdown, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

Tip

If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then select a dashboard.
3. In the upper right corner of the dashboard screen, choose the **Custom** dropdown.
4. In the upper right corner of the box that appears, select **UTC** or **Local timezone** from the dropdown.

Using Amazon CloudWatch metrics

Metrics are data about the performance of your systems. By default, many services provide free metrics for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). You can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. Amazon CloudWatch can load all the metrics in your account (both AWS resource metrics and application metrics that you provide) for search, graphing, and alarms.

Metric data is kept for 15 months, enabling you to view both up-to-the-minute data and historical data.

To graph metrics in the console, you can use CloudWatch Metrics Insights, a high-performance SQL query engine that you can use to identify trends and patterns within all your metrics in real time.

Contents

- [Query your metrics with CloudWatch Metrics Insights \(p. 55\)](#)
- [Viewing available metrics \(p. 71\)](#)
- [Searching for available metrics \(p. 74\)](#)
- [Getting statistics for a metric \(p. 75\)](#)
- [Graphing metrics \(p. 85\)](#)
- [Publishing custom metrics \(p. 94\)](#)
- [Using metric math \(p. 97\)](#)
- [Using search expressions in graphs \(p. 119\)](#)

Query your metrics with CloudWatch Metrics Insights

CloudWatch Metrics Insights is a powerful high-performance SQL query engine that you can use to query your metrics at scale. You can identify trends and patterns within all of your CloudWatch metrics in real time.

You can perform a Metrics Insights query using the CloudWatch console, or by using `GetMetricData` or `PutDashboard` using the AWS CLI or the AWS SDKs. Queries run in the console are free of charge. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

When you use the CloudWatch console, you can choose from a wide variety of pre-built sample queries and also create your own queries. When you create a query, you can use both a builder view that interactively prompts you and lets you browse through your existing metrics and dimensions to easily build a query, and an editor view to write queries from scratch, edit the queries you build in the builder view, and edit sample queries to customize them.

With Metrics Insights you can run queries at scale. By using the **GROUP BY** clause, you can flexibly group your metrics in real time into separate time series per specific dimension value, based on your use cases. Because Metrics Insights queries include an **ORDER BY** ability, you can use Metrics Insights to make "Top N" type queries that can scan millions of metrics in your account, and return the top 10 most CPU consuming instances, for example, to help you pinpoint and remedy latency issues in your applications.

Topics

- [Build your queries \(p. 56\)](#)

- Metrics Insights query components and syntax (p. 56)
- Using Metrics Insights queries with metric math (p. 62)
- SQL inference (p. 62)
- Metrics Insights sample queries (p. 63)
- Metrics Insights limits (p. 69)
- Metrics Insights glossary (p. 70)
- Troubleshooting Metrics Insights (p. 70)

Build your queries

You can run a CloudWatch Metrics Insights query using the CloudWatch console, the AWS CLI, or the AWS SDKs. Queries run in the console are free of charge. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

For more information about using the AWS SDKs to perform a Metrics Insights query, see [GetMetricData](#).

To run a query using the CloudWatch console, follow these steps:

To query your metrics using Metrics Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **Query** tab.
4. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. If you are satisfied with this query, you can skip the rest of this procedure. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.
5. To create your own query, you can use the **Builder** view, the **Editor** view, and also use a combination of both. You can switch between the two views anytime and see your work in progress in both views.

In the **Builder** view, you can browse and select the metric namespace, metric name, filter, group, and order options. For each of these options, the query builder offers you a list of possible selections from your environment to choose from.

In the **Editor** view, you can start writing your query. As you type, the editor offers suggestions based on the characters that you have typed so far.

6. When you are satisfied with your query, choose **Run**.
7. (Optional) Another way to edit a query that you have graphed is to choose the **Graphed metrics** tab and choose the edit icon next to the query formula in the **Details** column.
8. (Optional) To remove a query from the graph, choose **Graphed metrics** and choose the X icon at the right side of the row that displays your query.

Metrics Insights query components and syntax

CloudWatch Metrics Insights syntax is as follows.

```
SELECT FUNCTION(metricName)
FROM namespace | SCHEMA(...)
[ WHERE labelKey OPERATOR labelValue [AND ...] ]
[ GROUP BY labelKey [ , ... ] ]
[ ORDER BY FUNCTION() [ DESC | ASC ] ]
[ LIMIT number ]
```

The possible clauses in a Metrics Insights query are as follows. None of the keywords are case sensitive, but the identifiers such as the names of metrics, namespaces, and dimensions are case sensitive.

SELECT

Required. Specifies the function to use to aggregate observations in each time bucket (determined by the provided period). Also specifies the name of the metric to query.

The valid values for **FUNCTION** are **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM**.

- **AVG** calculates the average of the observations matched by the query.
- **COUNT** returns the count of the observations matched by the query.
- **MAX** returns the maximum value of the observations matched by the query.
- **MIN** returns the minimum value of the observations matched by the query.
- **SUM** calculates the sum of the observations matched by the query.

FROM

Required. Specifies the source of the metric. You can specify either the metric namespace that contains the metric that is to be queried, or a **SCHEMA** table function. Examples of metric namespaces include "**AWS/EC2**", "**AWS/Lambda**", and metric namespaces that you have created for your custom metrics.

Metric namespaces that include / or any other character that is not a letter, number, or underscore must be surrounded by double quotation marks. For more information, see [What needs quotation marks or escape characters? \(p. 59\)](#).

SCHEMA

An optional table function that can be used within a **FROM** clause. Use **SCHEMA** to scope down the query results to only the metrics that exactly match a list of dimensions, or to metrics that have no dimensions.

If you use a **SCHEMA** clause, it must contain at least one argument, and this first argument must be the metric namespace being queried. If you specify **SCHEMA** with only this namespace argument, the results are scoped down to only metrics that do not have any dimensions.

If you specify **SCHEMA** with additional arguments, the additional arguments after the namespace argument must be *label* keys. Label keys must be dimension names. If you specify one or more of these label keys, the results are scoped down to only those metrics that have that exact set of dimensions. The order of these label keys does not matter.

For example:

- **SELECT AVG(CPUUtilization) FROM "AWS/EC2"** matches all CPUUtilization metrics in the AWS/EC2 namespace, no matter their dimensions, and returns a single aggregated time series.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2")** matches only the CPUUtilization metrics in the AWS/EC2 namespace that do not have any dimensions defined.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId)** matches only the CPUUtilization metrics that were reported to CloudWatch with exactly one dimension, InstanceId.
- **SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)** matches only the RequestCount metrics that were reported to CloudWatch from AWS/ApplicationELB with exactly two dimensions, LoadBalancer and AvailabilityZone.

WHERE

Optional. Filters the results to only those metrics that match your specified expression using specific label values for one or more label keys. For example, **WHERE InstanceType = 'c3.4xlarge'** filters

the results to only `c3.4xlarge` instance types, and `WHERE InstanceType != 'c3.4xlarge'` filters the results to all instance types except `c3.4xlarge`.

Label values must always be enclosed with single quotation marks.

Supported operators

The `WHERE` clause supports the following operators:

- `=` Label value must match the specified string.
- `!=` Label value must not match the specified string.
- `AND` Both conditions that are specified must be true to match. You can use multiple `AND` keywords to specify two or more conditions.

GROUP BY

Optional. Groups the query results into multiple time series, each one corresponding to a different value for the specified label key or keys. For example, using `GROUP BY InstanceId` returns a different time series for each value of `InstanceId`. Using `GROUP BY ServiceName, Operation` creates a different time series for each possible combination of the values of `ServiceName` and `Operation`.

With a `GROUP BY` clause, by default the results are ordered in alphabetical ascending order, using the sequence of labels specified in the `GROUP BY` clause. To change the order of the results, add an `ORDER BY` clause to your query.

Note

If some of the matching metrics don't include a specific label key specified in the `GROUP BY` clause, a null group named `Other` is returned. For example, if you specify `GROUP BY ServiceName, Operation` and some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

ORDER BY

Optional. Specifies the order to use for the returned time series, if the query returns more than one time series. The order is based on the values found by the `FUNCTION` that you specify in the `ORDER BY` clause. The `FUNCTION` is used to calculate a single scalar value from each returned time series, and that value is used to determine the order.

You also specify whether to use ascending `ASC` or descending `DESC` order. If you omit this, the default is ascending `ASC`.

For example, adding an `ORDER BY MAX() DESC` clause orders the results by the maximum data point observed within the time range, in descending order: meaning that the time series that has the highest maximum data point is returned first.

The valid functions to use within an `ORDER BY` clause are `AVG()`, `COUNT()`, `MAX()`, `MIN()`, and `SUM()`.

If you use an `ORDER BY` clause with a `LIMIT` clause, the resulting query is a "Top N" query. `ORDER BY` is also useful for queries that might return a large number of metrics, because each query can return no more than 500 time series. If a query matches more than 500 time series, and you use an `ORDER BY` clause, the time series are sorted and then the 500 time series that come first in the sort order are the ones that are returned.

LIMIT

Optional. Limits the number of time series returned by the query to the value that you specify. The maximum value that you can specify is 500, and a query that does not specify a `LIMIT` can also return no more than 500 time series.

Using a **LIMIT** clause with an **ORDER BY** clause gives you a "Top N" query.

What needs quotation marks or escape characters?

In a query, label values must always be surrounded with single quotation marks. For example, **SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE AutoScalingGroupName = 'my-production-fleet'.**

Metric namespaces, metric names, and label keys that contain characters other than letters, numbers, and underscore (_) must be surrounded by double quote marks. For example, **SELECT MAX("My.Metric")**.

If one of these contains a double quotation mark or single quotation mark itself (such as `Bytes"Input"`), you must escape each quotation mark with a backslash, as in **SELECT AVG("Bytes \"Input\\\"")**.

If a metric namespace, metric name, or label key, contains a word that is a reserved keyword in Metrics Insights, these must also be enclosed in double quotation marks. For example, if you have a metric named `LIMIT`, you would use `SELECT AVG("LIMIT")`. It is also valid to enclose any namespace, metric name, or label in double quotation marks even if it does not include a reserved keyword.

For a complete list of reserved keywords, see [Reserved keywords \(p. 60\)](#).

Building a rich query step by step

This section illustrates building a full example that uses all possible clauses, step by step.

We start with the following query, which aggregates all of the Application Load Balancer RequestCount metrics that are collected with both the dimensions LoadBalancer and AvailabilityZone.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
```

Now, if we want to see metrics only from a specific load balancer, we can add a **WHERE** clause to limit the metrics returned to only those metrics where the value of the `LoadBalancer` dimension is `app/load-balancer-1`.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

The preceding query aggregates the RequestCount metrics from all Availability Zones for this load balancer into one time series. If we want to see different time series for each Availability Zone, we can add a **GROUP BY** clause.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

Next, we might want to order these results to see the highest values first. The following **ORDER BY** clause orders the time series in descending order, by the maximum value reported by each time series during the query time range:

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

```
ORDER BY MAX() DESC
```

Finally, if we are primarily interested in a "Top N" type of query, we can use a **LIMIT** clause. This final example limits the results to only the time series with the five highest **MAX** values.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
LIMIT 5
```

Reserved keywords

The following are reserved keywords in CloudWatch Metrics Insights. If any of these words are in a namespace, metric name, or label key in a query, you must enclose them in double quote marks. Reserved keywords are not case sensitive.

```
"ABORT" "ABORTSESSION" "ABS" "ABSOLUTE" "ACCESS" "ACCESSIBLE" "ACCESS_LOCK" "ACCOUNT"
"ACOS" "ACOSH" "ACTION" "ADD" "ADD_MONTHS"
"ADMIN" "AFTER" "AGGREGATE" "ALIAS" "ALL" "ALLOCATE" "ALLOW" "ALTER" "ALTERAND" "AMP"
"ANALYSE" "ANALYZE" "AND" "ANSIDATE" "ANY" "ARE" "ARRAY",
"ARRAY_AGG" "ARRAY_EXISTS" "ARRAY_MAX_CARDINALITY" "AS" "ASC" "ASENSITIVE" "ASIN" "ASINH"
"ASSERTION" "ASSOCIATE" "ASUTIME" "ASYMMETRIC" "AT",
"ATAN" "ATAN2" "ATANH" "ATOMIC" "AUDIT" "AUTHORIZATION" "AUX" "AUXILIARY" "AVE" "AVERAGE"
"AVG" "BACKUP" "BEFORE" "BEGIN" "BEGIN_FRAME" "BEGIN_PARTITION",
"BETWEEN" "BIGINT" "BINARY" "BIT" "BLOB" "BOOLEAN" "BOTH" "BREADTH" "BREAK" "BROWSE" "BT"
"BUFFERPOOL" "BULK" "BUT" "BY" "BYTE" "BYTEINT" "BYTES" "CALL",
"CALLED" "CAPTURE" "CARDINALITY" "CASCADE" "CASCADED" "CASE" "CASESPECIFIC" "CASE_N" "CAST"
"CATALOG" "CCSID" "CD" "CEIL" "CEILING" "CHANGE" "CHAR",
"CHAR2HEXINT" "CHARACTER" "CHARACTERS" "CHARACTER_LENGTH" "CHARS" "CHAR_LENGTH" "CHECK"
"CHECKPOINT" "CLASS" "CLASSIFIER" "CLOB" "CLONE" "CLOSE" "CLUSTER",
"CLUSTERED" "CM" "COALESCE" "COLLATE" "COLLATION" "COLLECT" "COLLECTION" "COLLID" "COLUMN"
"COLUMN_VALUE" "COMMENT" "COMMIT" "COMPLETION" "COMPRESS" "COMPUTE",
"CONCAT" "CONCURRENTLY" "CONDITION" "CONNECT" "CONNECTION" "CONSTRAINT" "CONSTRAINTS"
"CONSTRUCTOR" "CONTAINS" "CONTAINSTABLE" "CONTENT" "CONTINUE" "CONVERT",
"CONVERT_TABLE_HEADER" "COPY" "CORR" "CORRESPONDING" "COS" "COSH" "COUNT" "COVAR_POP"
"COVAR_SAMP" "CREATE" "CROSS" "CS" "CSUM" "CT" "CUBE" "CUME_DIST",
"CURRENT" "CURRENT_CATALOG" "CURRENT_DATE" "CURRENT_DEFAULT_TRANSFORM_GROUP"
"CURRENT_LC_CTYPE" "CURRENT_PATH" "CURRENT_ROLE" "CURRENT_ROW" "CURRENT_SCHEMA",
"CURRENT_SERVER" "CURRENT_TIME" "CURRENT_TIMESTAMP" "CURRENT_TIMEZONE"
"CURRENT_TRANSFORM_GROUP_FOR_TYPE" "CURRENT_USER" "CURRVAL" "CURSOR" "CV" "CYCLE" "DATA",
"DATABASE" "DATABASES" "DATABLOCKSIZE" "DATE" "DATEFORM" "DAY" "DAYS" "DAY_HOUR"
"DAY_MICROSECOND" "DAY_MINUTE" "DAY_SECOND" "DBCC" "DBINFO" "DEALLOCATE" "DEC",
"DECFLOAT" "DECIMAL" "DECLARE" "DEFAULT" "DEFERRABLE" "DEFERRED" "DEFINE" "DEGREES" "DEL"
"DELAYED" "DELETE" "DENSE_RANK" "DENY" "DEPTH" "DEREF" "DESC" "DESCRIBE",
"DESCRIPTOR" "DESTROY" "DESTRUCTOR" "DETERMINISTIC" "DIAGNOSTIC" "DIAGNOSTICS" "DICTIONARY"
"DISABLE" "DISABLED" "DISALLOW" "DISCONNECT" "DISK" "DISTINCT",
"DISTINCTROW" "DISTRIBUTED" "DIV" "DO" "DOCUMENT" "DOMAIN" "DOUBLE" "DROP" "DSSIZE" "DUAL"
"DUMP" "DYNAMIC" "EACH" "ECHO" "EDITPROC" "ELEMENT" "ELSE" "ELSEIF",
"EMPTY" "ENABLED" "ENCLOSED" "ENCODING" "ENCRYPTION" "END" "END-EXEC" "ENDING" "END_FRAME"
"END_PARTITION" "EQ" "EQUALS" "ERASE" "ERRLVL" "ERROR" "ERRORFILES",
"ERRORTABLES" "ESCAPE" "ESCAPED" "ET" "EVERY" "EXCEPT" "EXCEPTION" "EXCLUSIVE" "EXEC"
"EXECUTE" "EXISTS" "EXIT" "EXP" "EXPLAIN" "EXTERNAL" "EXTRACT" "FALLBACK"
"FALSE" "FASTEXPORT" "FENCED" "FETCH" "FIELDPROC" "FILE" "FILLFACTOR" "FILTER" "FINAL"
"FIRST" "FIRST_VALUE" "FLOAT" "FLOAT4" "FLOAT8" "FLOOR"
"FOR" "FORCE" "FOREIGN" "FORMAT" "FOUND" "FRAME_ROW" "FREE" "FREESPACE" "FREETEXT"
"FREETEXTTABLE" "FREEZE" "FROM" "FULL" "FULLTEXT" "FUNCTION"
"FUSION" "GE" "GENERAL" "GENERATED" "GET" "GIVE" "GLOBAL" "GO" "GOTO" "GRANT" "GRAPHIC"
"GROUP" "GROUPING" "GROUPS" "GT" "HANDLER" "HASH"
"HASHAMP" "HASHBAKAMP" "HASHBUCKET" "HASHROW" "HAVING" "HELP" "HIGH_PRIORITY" "HOLD"
"HOLDLOCK" "HOUR" "HOURS" "HOUR_MICROSECOND" "HOUR_MINUTE"
```

```

"HOUR_SECOND" "IDENTIFIED" "IDENTITY" "IDENTITYCOL" "IDENTITY_INSERT" "IF" "IGNORE" "ILIKE"
"IMMEDIATE" "IN" "INCLUSIVE" "INCONSISTENT" "INCREMENT"
"INDEX" "INDICATOR" "INFILE" "INHERIT" "INITIAL" "INITIALIZE" "INITIALLY" "INITIATE"
"INNER" "INOUT" "INPUT" "INS" "INSENSITIVE" "INSERT" "INSTEAD"
"INT" "INT1" "INT2" "INT3" "INT4" "INT8" "INTEGER" "INTEGERDATE" "INTERSECT" "INTERSECTION"
"INTERVAL" "INTO" "IO_AFTER_GTIDS" "IO_BEFORE_GTIDS"
"IS" "ISNULL" "ISOBID" "ISOLATION" "ITERATE" "JAR" "JOIN" "JOURNAL" "JSON_ARRAY"
"JSON_ARRAYAGG" "JSON_EXISTS" "JSON_OBJECT" "JSON_OBJECTAGG"
"JSON_QUERY" "JSON_TABLE" "JSON_TABLE_PRIMITIVE" "JSON_VALUE" "KEEP" "KEY" "KEYS" "KILL"
"KURTOSIS" "LABEL" "LAG" "LANGUAGE" "LARGE" "LAST"
"LAST_VALUE" "LATERAL" "LC_CTYPE" "LE" "LEAD" "LEADING" "LEAVE" "LEFT" "LESS" "LEVEL"
"LIKE" "LIKE_REGEX" "LIMIT" "LINEAR" "LINENO" "LINES"
"LISTAGG" "LN" "LOAD" "LOADING" "LOCAL" "LOCALE" "LOCALTIME" "LOCALTIMESTAMP" "LOCATOR"
"LOCATORS" "LOCK" "LOCKING" "LOCKMAX" "LOCKSIZE" "LOG"
"LOG10" "LOGGING" "LOGON" "LONG" "LONGBLOB" "LONGTEXT" "LOOP" "LOWER" "LOW_PRIORITY" "LT"
"MACRO" "MAINTAINED" "MAP" "MASTER_BIND"
"MASTER_SSL_VERIFY_SERVER_CERT" "MATCH" "MATCHES" "MATCH_NUMBER" "MATCH_RECOGNIZE"
"MATIALIZED" "MAVG" "MAX" "MAXEXTENTS" "MAXIMUM" "MAXVALUE"
"MCCHARACTERS" "MDIFF" "MEDIUMBLOB" "MEDIUMINT" "MEDIUMTEXT" "MEMBER" "MERGE" "METHOD"
"MICROSECOND" "MICROSECONDS" "MIDDLEINT" "MIN" "MINDEX"
"MINIMUM" "MINUS" "MINUTE" "MINUTES" "MINUTE_MICROSECOND" "MINUTE_SECOND" "MLINREG" "MLOAD"
"MLSLABEL" "MOD" "MODE" "MODIFIES" "MODIFY"
"MODULE" "MONITOR" "MONRESOURCE" "MONSESSION" "MONTH" "MONTHS" "MSUBSTR" "MSUM" "MULTISET"
"NAMED" "NAMES" "NATIONAL" "NATURAL" "NCHAR" "NCLOB"
"NE" "NESTED_TABLE_ID" "NEW" "NEW_TABLE" "NEXT" "NEXTVAL" "NO" "NOAUDIT" "NOCHECK"
"NOCOMPRESS" "NONCLUSTERED" "NONE" "NORMALIZE" "NOT" "NOTNULL"
"NOWAIT" "NO_WRITE_TO_BINLOG" "NTH_VALUE" "NTILE" "NULL" "NULLIF" "NULLIFZERO" "NULLS"
"NUMBER" "NUMERIC" "NUMPARTS" "OBID" "OBJECT" "OBJECTS"
"Occurrences_REGEX" "OCTET_LENGTH" "OF" "OFF" "OFFLINE" "OFFSET" "OFFSETS" "OLD"
"OLD_TABLE" "OMIT" "ON" "ONE" "ONLINE" "ONLY" "OPEN" "OPENDATASOURCE"
"OPENQUERY" "OPENROWSET" "OPENXML" "OPERATION" "OPTIMIZATION" "OPTIMIZE" "OPTIMIZER_COSTS"
"OPTION" "OPTIONALLY" "OR" "ORDER" "ORDINALITY" "ORGANIZATION"
"OUT" "OUTER" "OUTFILE" "OUTPUT" "OVER" "OVERLAPS" "OVERLAY" "OVERRIDE" "PACKAGE" "PAD"
"PADDED" "PARAMETER" "PARAMETERS" "PART" "PARTIAL" "PARTITION"
"PARTITIONED" "PARTITIONING" "PASSWORD" "PATH" "PATTERN" "PCTFREE" "PER" "PERCENT"
"PERCENTILE" "PERCENTILE_CONT" "PERCENTILE_DISC" "PERCENT_RANK" "PERIOD" "PERM"
"PERMANENT" "PIECESIZE" "PIVOT" "PLACING" "PLAN" "PORTION" "POSITION" "POSITION_REGEX"
"POSTFIX" "POWER" "PRECEDES" "PRECISION" "PREFIX" "PREORDER"
"PREPARE" "PRESERVE" "PREVVAL" "PRIMARY" "PRINT" "PRIOR" "PRIQTY" "PRIVATE" "PRIVILEGES"
"PROC" "PROCEDURE" "PROFILE" "PROGRAM" "PROPORTIONAL"
"PROTECTION" "PSID" "PTF" "PUBLIC" "PURGE" "QUALIFIED" "QUALIFY" "QUANTILE" "QUERY"
"QUERYNO" "RADIAN" "RAISERROR" "RANDOM" "RANGE" "RANGE_N" "RANK"
"RAW" "READ" "READS" "READTEXT" "READ_WRITE" "REAL" "RECONFIGURE" "RECURSIVE" "REF"
"REFERENCES" "REFERENCING" "REFRESH" "REGEXP" "REGR_AVGX" "REGR_AVGY"
"REGR_COUNT" "REGR_INTERCEPT" "REGR_R2" "REGR_SLOPE" "REGR_SXX" "REGR_SXY" "REGR_SYY"
"RELATIVE" "RELEASE" "RENAME" "REPEAT" "REPLACE" "REPLICATION"
"REPOVERRIDE" "REQUEST" "REQUIRE" "RESIGNAL" "RESOURCE" "RESTART" "RESTORE" "RESTRICT"
"RESULT" "RESULT_SET_LOCATOR" "RESUME" "RET" "RETRIEVE" "RETURN"
"RETURNING" "RETURNS" "REVALIDATE" "REVERT" "REVOKE" "RIGHT" "RIGHTS" "RLIKE" "ROLE"
"ROLLBACK" "ROLLFORWARD" "ROLLUP" "ROUND_CEILING" "ROUND_DOWN"
"ROUND_FLOOR" "ROUND_HALF_DOWN" "ROUND_HALF_EVEN" "ROUND_HALF_UP" "ROUND_UP" "ROUTINE"
"ROW" "ROWCOUNT" "ROWGUIDCOL" "ROWID" "ROWNUM" "ROWS" "ROWSET"
"ROW_NUMBER" "RULE" "RUN" "RUNNING" "SAMPLE" "SAMPLEID" "SAVE" "SAVEPOINT" "SCHEMA"
"SCHEMAS" "SCOPE" "SCRATCHPAD" "SCROLL" "SEARCH" "SECOND" "SECONDS"
"SECOND_MICROSECOND" "SECQTY" "SECTION" "SECURITY" "SECURITYAUDIT" "SEEK" "SEL" "SELECT"
"SEMANTICKEYPHRASETABLE" "SEMANTICSIMILARITYDETAILSTABLE"
"SEMANTICSIMILARITYTABLE" "SENSITIVE" "SEPARATOR" "SEQUENCE" "SESSION" "SESSION_USER" "SET"
"SETRESRATE" "SETS" "SETSESSRATE" "SETUSER" "SHARE" "SHOW"
"SHUTDOWN" "SIGNAL" "SIMILAR" "SIMPLE" "SIN" "SINH" "SIZE" "SKEW" "SKIP" "SMALLINT" "SOME"
"SOUNDEX" "SOURCE" "SPACE" "SPATIAL" "SPECIFIC" "SPECIFICTYPE"
"SPOOL" "SQL" "SOLEXCEPTION" "SQLSTATE" "SQLTEXT" "SQLWARNING" "SQL_BIG_RESULT"
"SQL_CALC_FOUND_ROWS" "SQL_SMALL_RESULT" "SQR" "SS" "SSL" "STANDARD"
"START" "STARTING" "STARTUP" "STAT" "STATE" "STATEMENT" "STATIC" "STATISTICS" "STAY"
"STDDEV_POP" "STDDEV_SAMP" "STEPINFO" "STOGROUP" "STORED" "STORES"
"STRAIGHT_JOIN" "STRING_CS" "STRUCTURE" "STYLE" "SUBMULTISET" "SUBSCRIBER" "SUBSET"
"SUBSTR" "SUBSTRING" "SUBSTRING_REGEX" "SUCCEEDS" "SUCCESSFUL"

```

```
"SUM" "SUMMARY" "SUSPEND" "SYMMETRIC" "SYNONYM" "SYSDATE" "SYSTEM" "SYSTEM_TIME"  
"SYSTEM_USER" "SYSTIMESTAMP" "TABLE" "TABLESAMPLE" "TABLESPACE" "TAN"  
"TANH" "TBL_CS" "TEMPORARY" "TERMINATE" "TERMINATED" "TEXTSIZE" "THAN" "THEN" "THRESHOLD"  
"TIME" "TIMESTAMP" "TIMEZONE_HOUR" "TIMEZONE_MINUTE" "TINYBLOB"  
"TINYINT" "TINYTEXT" "TITLE" "TO" "TOP" "TRACE" "TRAILING" "TRAN" "TRANSACTION" "TRANSLATE"  
"TRANSLATE_CHK" "TRANSLATE_REGEX" "TRANSLATION" "TREAT"  
"TRIGGER" "TRIM" "TRIM_ARRAY" "TRUE" "TRUNCATE" "TRY_CONVERT" "TSEQUAL" "TYPE" "UC"  
"UESCAPE" "UID" "UNDEFINED" "UNDER" "UNDO" "UNION" "UNIQUE"  
"UNKNOWN" "UNLOCK" "UNNEST" "UNPIVOT" "UNSIGNED" "UNTIL" "UPD" "UPDATE" "UPDATETEXT"  
"UPPER" "UPPERCASE" "USAGE" "USE" "USER" "USING" "UTC_DATE"  
"UTC_TIME" "UTC_TIMESTAMP" "VALIDATE" "VALIDPROC" "VALUE" "VALUES" "VALUE_OF" "VARBINARY"  
"VARBYTE" "VARCHAR" "VARCHAR2" "VARCHARACTER" "VARGRAPHIC"  
"VARIABLE" "VARIADIC" "VARIANT" "VARYING" "VAR_POP" "VAR_SAMP" "VCAT" "VERBOSE"  
"VERSIONING" "VIEW" "VIRTUAL" "VOLATILE" "VOLUMES" "WAIT" "WAITFOR"  
"WHEN" "WHENEVER" "WHERE" "WHILE" "WIDTH_BUCKET" "WINDOW" "WITH" "WITHIN" "WITHIN_GROUP"  
"WITHOUT" "WLM" "WORK" "WRITE" "WRITETEXT" "XMLCAST" "XMLEXISTS"  
"XMLNAMESPACES" "XOR" "YEAR" "YEARS" "YEAR_MONTH" "ZEROFILL" "ZEROIFNULL" "ZONE"
```

Using Metrics Insights queries with metric math

You can use a Metrics Insights query as an input to a metric math function. For more information about metric math, see [Using metric math \(p. 97\)](#).

A Metrics Insights query that does not include a **GROUP BY** clause returns a single time series. Therefore, its returned results can be used with any metric math function that takes a single time series as input.

A Metrics Insights query that includes a **GROUP BY** clause returns multiple time series. Therefore, its returned results can be used with any metric math function that takes an array of time series as input.

For example, the following query returns the total number of bytes downloaded for each bucket in the Region, as an array of time series:

```
SELECT SUM(BytesDownloaded)  
FROM SCHEMA("AWS/S3", BucketName, FilterId)  
WHERE FilterId = 'EntireBucket'  
GROUP BY BucketName
```

On a graph in the console or in a [GetMetricData](#) operation, the results of this query are `q1`. This query returns the result in bytes, so if you want to see the result as MB instead, you can use the following math function:

```
q1/1024/1024
```

SQL inference

CloudWatch Metrics Insights uses several mechanisms to infer the intention of a given SQL query.

Topics

- [Time bucketing \(p. 62\)](#)
- [Fields projection \(p. 63\)](#)
- [ORDER BY global aggregation \(p. 63\)](#)

Time bucketing

Time series data points resulting from a query are rolled up into time buckets based on the requested period. To aggregate values in standard SQL, an explicit GROUP BY clause must be defined to collect all

the observations of a given period together. Because this is the standard way to query time series data, CloudWatch Metrics Insights infers time bucketing without the need to express an explicit **GROUP BY** clause.

For example, when a query is performed with a period of one minute, all the observations belonging to that minute until the next (excluded) are rolled up to the start time of the time bucket. This makes Metrics Insights SQL statements more concise and less verbose.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

The previous query returns a single time series (timestamp-value pairs), representing the average CPU utilization of all Amazon EC2 instances. Assuming the requested period is one minute, each data point returned represents the average of all observations measured within a specific one-minute interval (start time inclusive, end time exclusive). The timestamp related to the specific data point is the start time of the bucket

Fields projection

Metrics Insights queries always return the timestamp projection. You don't need to specify a timestamp column in the **SELECT** clause to get the timestamp of each corresponding data point value. For details about how timestamp is calculated, see [Time bucketing \(p. 62\)](#).

When using **GROUP BY**, each group name is also inferred and projected in the result, so that you can group the returned time series.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

The previous query returns a time series for each Amazon EC2 instance. Each time series is labelled after the value of the instance ID.

ORDER BY global aggregation

When using **ORDER BY, FUNCTION()** infers which aggregate function that you want to order by (the data point values of the queried metrics). The aggregate operation is performed across all the matched data points of each individual time series across the queried time window.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX()
LIMIT 10
```

The previous query returns the CPU utilization for each Amazon EC2 instance, limiting the result set to 10 entries. The results are ordered based on the maximum value of the individual time series within the requested time window. The **ORDER BY** clause is applied before **LIMIT**, so that the ordering is calculated against more than 10 time series.

Metrics Insights sample queries

This section contains examples of useful CloudWatch Metrics Insights queries that you can copy and use directly or copy and modify in query editor. Some of these examples are already available in the console, and you can access them by choosing **Add query** in the **Metrics** view.

Application Load Balancer examples

Total requests across all load balancers

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
```

Top 10 most active load balancers

```
SELECT MAX(ActiveConnectionCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
GROUP BY LoadBalancer
ORDER BY SUM() DESC
LIMIT 10
```

AWS API usage examples

Top 20 AWS APIs by the number of calls in your account

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API'
GROUP BY Service, Resource
ORDER BY COUNT() DESC
LIMIT 20
```

CloudWatch APIs sorted by calls

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API' AND Service = 'CloudWatch'
GROUP BY Resource
ORDER BY COUNT() DESC
```

DynamoDB examples

Top 10 tables by consumed reads

```
SELECT SUM(ProvisionedWriteCapacityUnits)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by returned bytes

```
SELECT SUM(ReturnedBytes)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by user errors

```
SELECT SUM(UserErrors)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
```

```
| ORDER BY MAX() DESC LIMIT 10
```

Amazon Elastic Block Store examples

Top 10 Amazon EBS volumes by bytes written

```
SELECT SUM(VolumeWriteBytes)
FROM SCHEMA("AWS/EBS", VolumeId)
GROUP BY VolumeId
ORDER BY SUM() DESC
LIMIT 10
```

Average Amazon EBS volume write time

```
SELECT AVG(VolumeTotalWriteTime)
FROM SCHEMA("AWS/EBS", VolumeId)
```

Amazon EC2 examples

CPU utilization of EC2 instances sorted by highest

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY AVG() DESC
```

Average CPU utilization across the entire fleet

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

Top 10 instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX() DESC
LIMIT 10
```

In this case, the CloudWatch agent is collecting a CPUUtilization metric per application. This query filters the average of this metric for a specific application name.

```
SELECT AVG(CPUUtilization)
FROM "AWS/CWAgent"
WHERE ApplicationName = 'eCommerce'
```

Amazon Elastic Container Service examples

Average CPU utilization across all ECS clusters

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
```

Top 10 clusters by memory utilization

```
SELECT AVG(MemoryUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by CPU utilization

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by running tasks (Container Insights)

```
SELECT AVG(RunningTaskCount)
FROM SCHEMA("ECS/ContainerInsights", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Amazon Elastic Kubernetes Service Container Insights examples

Average CPU utilization across all EKS clusters

```
SELECT AVG(pod_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
```

Top 10 clusters by node CPU utilization

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 clusters by pod memory utilization

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 nodes by CPU utilization

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, NodeName)
GROUP BY ClusterName, NodeName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 pods by memory utilization

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, PodName)
GROUP BY ClusterName, PodName
ORDER BY AVG() DESC LIMIT 10
```

EventBridge examples

Top 10 rules by invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by failed invocations

```
SELECT SUM(FailedInvocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by matched rules

```
SELECT SUM(MatchedEvents)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Kinesis examples

Top 10 streams by bytes written

```
SELECT SUM("PutRecords.Bytes")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY SUM() DESC LIMIT 10
```

Top 10 streams by earliest items in the stream

```
SELECT MAX("GetRecords.IteratorAgeMilliseconds")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY MAX() DESC LIMIT 10
```

Lambda examples

Lambda functions ordered by number of invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
```

Top 10 Lambda functions by longest runtime

```
SELECT AVG(Duration)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Lambda functions by error count

```
SELECT SUM(Errors)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
LIMIT 10
```

CloudWatch Logs examples

Top 10 log groups by incoming events

```
SELECT SUM(IncomingLogEvents)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Top 10 log groups by written bytes

```
SELECT SUM(IncomingBytes)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Amazon RDS examples

Top 10 Amazon RDS instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/RDS", DBInstanceIdentifier)
GROUP BY DBInstanceIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Amazon RDS clusters by writes

```
SELECT SUM(WriteIOPS)
FROM SCHEMA("AWS/RDS", DBClusterIdentifier)
GROUP BY DBClusterIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Amazon Simple Storage Service examples

Average latency by bucket

```
SELECT AVG(TotalRequestLatency)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY AVG() DESC
```

Top 10 buckets by bytes downloaded

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
```

```
GROUP BY BucketName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon Simple Notification Service examples

Total messages published by SNS topics

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```

Top 10 topics by messages published

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 topics by message delivery failures

```
SELECT SUM(NumberOfNotificationsFailed)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon SQS examples

Top 10 queues by age of number of visible messages

```
SELECT AVG(ApproximateNumberOfMessagesVisible)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 most active queues

```
SELECT SUM(NumberOfMessagesSent)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 queues by age of earliest message

```
SELECT AVG(ApproximateAgeOfOldestMessage)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Metrics Insights limits

CloudWatch Metrics Insights currently has the following limits:

- Currently, you can query only the most recent three hours of data.
- A single query can process no more than 10,000 metrics. This means that if the **SELECT**, **FROM**, and **WHERE** clauses match more than 10,000 metrics, the query only processes the first 10,000 of these metrics that it finds.
- A single query can return no more than 500 time series. This means that if the query would return more than 500 metrics, not all metrics will be returned in the query results. If you use an **ORDER BY** clause, then all the metrics being processed are sorted, and the 500 that have the highest or lowest values according to your **ORDER BY** clause are returned.

If you do not include an **ORDER BY** clause, you can't control which 500 matching metrics are returned.

- Each [GetMetricData](#) operation can have only one query, but you can have multiple widgets in a dashboard that each include a query.

Metrics Insights glossary

label

In Metrics Insights, a label is a key-value pair that is used to scope a query to return a particular set of data, or to define criteria by which query results are to be separated into separate time series. A label key is similar to a column name in SQL. Currently, labels must be CloudWatch metric dimensions.

observation

An observation is a value recorded for a given metric at a given time.

Troubleshooting Metrics Insights

The results include "Other," but I don't have this as a dimension

This means that the query includes a **GROUP BY** clause that specifies a label key that is not used in some of the metrics that are returned by the query. In this case, a null group named **Other** is returned. The metrics that do not include that label key are probably aggregated metrics that return values aggregated across all values of that label key.

For example, suppose we have the following query:

```
SELECT AVG(Faults)
FROM MyCustomNamespace
GROUP BY Operation, ServiceName
```

If some of the returned metrics don't include **ServiceName** as a dimension, then those metrics are displayed as having **Other** as the value for **ServiceName**.

To prevent seeing "Other" in your results, use **SCHEMA** in your **FROM** clause, as in the following example:

```
SELECT AVG(Faults)
FROM SCHEMA(MyCustomNamespace, Operation)
GROUP BY Operation, ServiceName
```

This limits the returned results to only the metrics that have both the **Operation** and **ServiceName** dimensions.

The oldest timestamp in my graph has a lower metric value than the others

CloudWatch Metrics Insights currently supports the latest three hours of data only. When you graph with a period larger than one minute, for example five minutes or one hour, there could be cases where the oldest data point differs from the expected value. This is because the Metrics Insights queries return only the most recent 3 hours of data, so the oldest datapoint, being older than 3 hours, accounts only for observations that have been measured within the last three hours boundary.

Viewing available metrics

Metrics are grouped first by namespace, and then by the various dimension combinations within each namespace. For example, you can view all EC2 metrics, EC2 metrics grouped by instance, or EC2 metrics grouped by Auto Scaling group.

Only the AWS services that you're using send metrics to Amazon CloudWatch.

For a list of AWS services that send metrics to CloudWatch, see [AWS services that publish CloudWatch metrics \(p. 746\)](#). From this page, you can also see the metrics and dimensions that are published by each of those services.

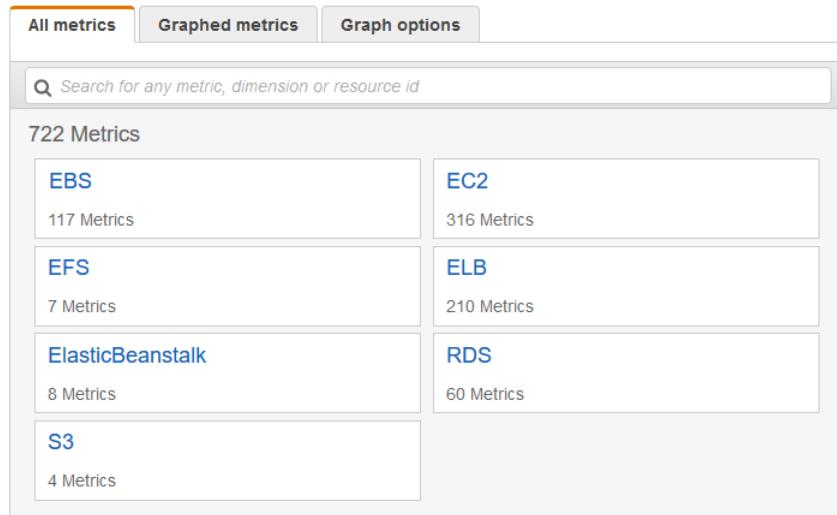
Note

Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a `list-metrics` command. The best way to retrieve these metrics is with the `get-metric-data` or `get-metric-statistics` commands in the AWS CLI.

If the old metric you want to view has a current metric with similar dimensions, you can view that current similar metric and then choose the **Source** tab, and change the metric name and dimension fields to the ones that you want, and also change the time range to a time when the metric was being reported.

To view available metrics by namespace and dimension using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select a metric namespace (for example, **EC2**).



4. Select a metric dimension (for example, **Per-Instance Metrics**).

The screenshot shows the CloudWatch Metrics console interface. At the top, there are three tabs: 'All metrics' (selected), 'Graphed metrics', and 'Graph options'. Below the tabs, the URL is shown as 'All > EC2'. A search bar contains the placeholder 'Search for any metric, dimension or resource id'. The main area displays '316 Metrics'. Under the heading 'By Auto Scaling Group', there are 124 Metrics. Under the heading 'Per-Instance Metrics', there are 192 Metrics.

5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:

- a. To sort the table, use the column heading.
- b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
- c. To filter by resource, choose the resource ID and then choose **Add to search**.
- d. To filter by metric, choose the metric name and then choose **Add to search**.

The screenshot shows the 'Per-Instance Metrics' table within the CloudWatch Metrics console. The columns are 'Instance Name (192)', 'InstanceId', and 'Metric Name'. The 'Metric Name' column for the first row ('my-instance') has a context menu open, with the 'Add to search' option highlighted. Other options in the menu include 'Search for this only', 'Add to graph', 'Graph this metric only', 'Graph all search results', and 'Jump to resource'.

| Instance Name (192) | InstanceId | Metric Name |
|---------------------|------------|-------------------|
| my-instance | i-abbc12a7 | CPUUtilization |
| my-instance | | DiskReadBytes |
| my-instance | | DiskReadOps |
| my-instance | | DiskWriteBytes |
| my-instance | | DiskWriteOps |
| my-instance | | NetworkIn |
| my-instance | | NetworkOut |
| my-instance | i-abbc12a7 | NetworkPacketsIn |
| my-instance | i-abbc12a7 | NetworkPacketsOut |

6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

To view available metrics by namespace, dimension, or metric using the AWS CLI

Use the [list-metrics](#) command to list CloudWatch metrics. For a list of the namespaces, metrics, and dimensions for all services that publish metrics, see [AWS services that publish CloudWatch metrics \(p. 746\)](#).

The following example specifies the AWS/EC2 namespace to view all the metrics for Amazon EC2.

```
aws cloudwatch list-metrics --namespace AWS/EC2
```

The following is example output.

```
{  
    "Metrics" : [  
        ...  
        {  
            "Namespace": "AWS/EC2",  
            "Dimensions": [  
                {  
                    "Name": "InstanceId",  
                    "Value": "i-1234567890abcdef0"  
                }  
            ],  
            "MetricName": "NetworkOut"  
        },  
        {  
            "Namespace": "AWS/EC2",  
            "Dimensions": [  
                {  
                    "Name": "InstanceId",  
                    "Value": "i-1234567890abcdef0"  
                }  
            ],  
            "MetricName": "CPUUtilization"  
        },  
        {  
            "Namespace": "AWS/EC2",  
            "Dimensions": [  
                {  
                    "Name": "InstanceId",  
                    "Value": "i-1234567890abcdef0"  
                }  
            ],  
            "MetricName": "NetworkIn"  
        },  
        ...  
    ]  
}
```

To list all the available metrics for a specified resource

The following example specifies the AWS/EC2 namespace and the `InstanceId` dimension to view the results for the specified instance only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --dimensions  
    Name=InstanceId,Value=i-1234567890abcdef0
```

To list a metric for all resources

The following example specifies the AWS/EC2 namespace and a metric name to view the results for the specified metric only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --metric-name CPUUtilization
```

Searching for available metrics

You can search within all of the metrics in your account using targeted search terms. Metrics are returned that have matching results within their namespace, metric name, or dimensions.

Note

Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a [list-metrics](#) command. The best way to retrieve these metrics is with the [get-metric-data](#) or [get-metric-statistics](#) commands in the AWS CLI.

To search for available metrics in CloudWatch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the search field on the **All metrics** tab, enter a search term, such as a metric name, namespace, dimension name or value, or resource name. This shows you all of the namespaces with metrics with this search term.

For example, if you search for **volume**, this shows the namespaces that contain metrics with this term in their name.

For more information on search, see [Using search expressions in graphs \(p. 119\)](#)

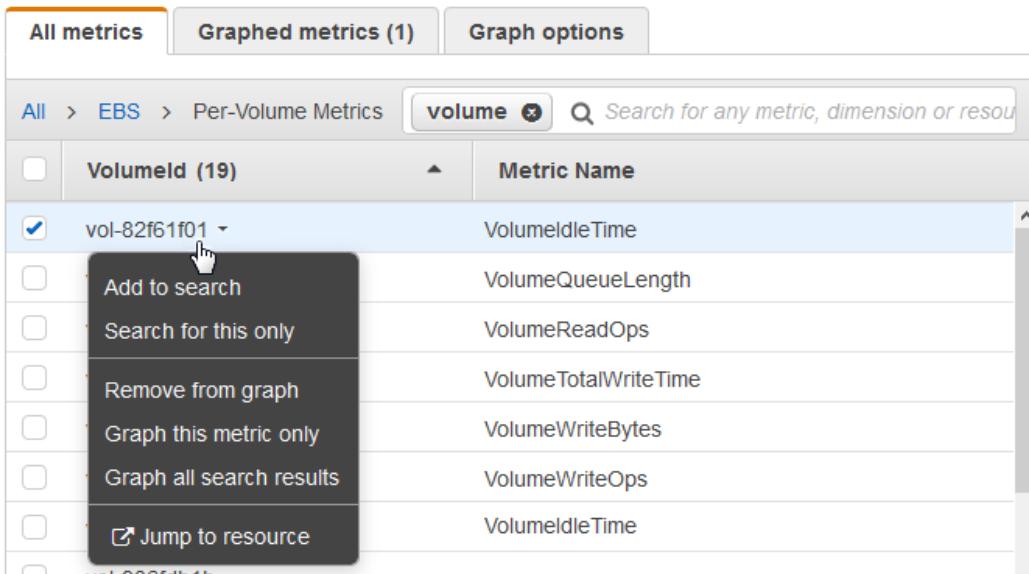
4. To graph all the search results, choose **Graph search**

or

Select a namespace to view the metrics from that namespace. You can then do the following:

- a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
- b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
- c. To view one of the resources on its console, choose the resource ID and then choose **Jump to resource**.
- d. To view help for a metric, select the metric name and choose **What is this?**.

The selected metrics appear on the graph.



5. (Optional) Select one of the buttons in the search bar to edit that part of the search term.

Getting statistics for a metric

CloudWatch statistics definitions

Statistics are metric data aggregations over specified periods of time. When you graph or retrieve the statistics for a metric, you specify the *Period* of time, such as five minutes, to use to calculate each statistical value. For example, if the **Period** is five minutes, the **Sum** is the sum of all sample values collected during the five-minute period, while the **Minimum** is the lowest value collected during the five-minute period.

CloudWatch supports the following statistics for metrics.

- **SampleCount** is the number of data points during the period.
- **Sum** is sum of the values of the all data points collected during the period.
- **Average** is the value of Sum/SampleCount during the specified period.
- **Minimum** is the lowest value observed during the specified period.
- **Maximum** is the highest value observed during the specified period.
- **Percentile (p)** indicates the relative standing of a value in a dataset. For example, **p95** is the 95th percentile and means that 95 percent of the data within the period is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.
- **Trimmed mean (TM)** is the mean of all values that are between two specified boundaries. Values outside of the boundaries are ignored when the mean is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. The numbers can be absolute values or percentages. For example, **tm90** calculates the average after removing the 10% of data points with the highest values. **TM(2%:98%)** calculates the average after removing the 2% lowest data points and the 2% highest data points. **TM(150:1000)** calculates the average after removing all data points that are lower than or equal to 150, or higher than 1000.
- **Interquartile mean (IQM)** is the trimmed mean of the *interquartile range*, or the middle 50% of values. It is equivalent to **TM(25%:75%)**.

- **Winsorized mean (WM)** is similar to trimmed mean. However, with winsorized mean, the values that are outside the boundary are not ignored, but instead are considered to be equal to the value at the edge of the appropriate boundary. After this normalization, the average is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. For example, **wm98** calculates the average while treating the 2% of the highest values to be equal to the value at the 98th percentile. **WM(10%:90%)** calculates the average while treating the highest 10% of data points to be the value of the 90% boundary, and treating the lowest 10% of data points to be the value of the 10% boundary.
- **Percentile rank (PR)** is the percentage of values that meet a fixed threshold. For example, **PR(:300)** returns the percentage of data points that have a value of 300 or less. **PR(100:2000)** returns the percentage of data points that have a value between 100 and 2000.
- **Trimmed count (TC)** is the number of data points in the chosen range for a trimmed mean statistic. For example, **tc90** returns the number of data points not including any data points that fall in the highest 10% of the values. **TC(0.005:0.030)** returns the number of data points with values between 0.005 (exclusive) and 0.030 (inclusive).
- **Trimmed sum (TS)** is the sum of the values of data points in a chosen range for a trimmed mean statistic. It is equivalent to (Trimmed Mean) * (Trimmed count). For example, **ts90** returns the sum of the data points not including any data points that fall in the highest 10% of the values. **TS(80%:)** returns the sum of the data point values, not including any data points with values in the lowest 80% of the range of values.

Note

For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, if you define two boundaries as fixed values instead of percentages, the calculation includes values equal to the higher boundary, but does not include values equal to the lower boundary.

Syntax

For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, the following syntax rules apply:

- Using parentheses with one or two numbers with percent signs defines the boundaries to use as the values in the data set that fall in between the two percentiles that you specify. For example, **TM(10%:90%)** uses only the values between the 10th and 90th percentiles. **TM(:95%)** uses the values from the lowest end of the data set up to the 95th percentile, ignoring the 5% of data points with the highest values.
- Using parenthesis with one or two numbers without percent signs defines the boundaries to use as the values in the data set that fall in between the explicit values that you specify. For example, **TC(80:500)** uses only the values that are between 80 (exclusive) and 500 (inclusive). **TC(:0.5)** uses only the values that equal 0.5 or are lower.
- Using one number without parentheses calculates using percentages, ignoring data points that are higher than the specified percentile. For example, **tm99** calculates the mean while ignoring the 1% of the data points with the highest value. It is the same as **TM(:99%)**.
- Trimmed mean, Trimmed Count, Trimmed Sum, and Winsorized Mean can all be abbreviated using uppercase letters when specifying a range, such as **TM(5%:95%)**, **TM(100:200)**, or **TM(:95%)**. They can only be abbreviated using lowercase letters when you specifying only one number, such as **tm99**.

Statistics use cases

- **Trimmed mean** is most useful for metrics with a large sample size, such as webpage latency. For example, **tm99** disregards extreme high outliers that could result from network problems or human errors, to give a more accurate number for the average latency of typical requests. Similarly, **TM(10%:)** disregards the lowest 10% of latency values, such as those resulting from cache hits. And **TM(10%:99%)** excludes both of these types of outliers.

- It is a good idea to keep watch on trimmed count whenever you are using trimmed mean, to make sure that the number of values being used in your trimmed mean calculations are enough to be statistically significant.
- Percentile rank enables you to put values into "bins" of ranges, and you can use this to manually create a histogram. To do this, break your values down into various bins, such as **PR(:1)**, **PR(1:5)**, **PR(5:10)**, and **PR(10:)**. Put each of these bins into a visualization as bar charts, and you have a histogram.

Percentiles versus trimmed mean

A percentile such as **p99** and a trimmed mean such as **tm99** measure similar, but not identical values. Both **p99** and **tm99** ignore the 1% of the data points with the highest values, which are considered outliers. After that, **p99** is the maximum value of the remaining 99%, while **tm99** is the *average* of the remaining 99%. If you are looking at the latency of web requests, **p99** tells you the worst customer experience, ignoring outliers, while **tm99** tells you the average customer experience, ignoring outliers.

Trimmed mean is a good latency statistic to watch if you are looking to optimize your customer experience. For alarming on latency, we recommend to use a percentile statistic, to receive alerts early if there is an issue that leads to a partial loss of service.

Requirements to use percentiles, trimmed mean, and some other statistics

CloudWatch needs raw data points to calculate the following statistics:

- Percentiles
- Trimmed mean
- Interquartile mean
- Winsorized mean
- Trimmed sum
- Trimmed count
- Percentile rank

If you publish data for a custom statistics using a statistic set instead of raw data, you can retrieve these types of statistics for this data only if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following AWS services include metrics that support these types of statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

Additionally, these type of statistics are not available for metrics when any of the metric values are negative numbers.

The following examples show you how to get statistics for the CloudWatch metrics for your resources, such as your EC2 instances.

Examples

- [Getting statistics for a specific resource \(p. 78\)](#)
- [Aggregating statistics across resources \(p. 81\)](#)
- [Aggregating statistics by Auto Scaling group \(p. 82\)](#)
- [Aggregating statistics by Amazon Machine Image \(AMI\) \(p. 84\)](#)

Getting statistics for a specific resource

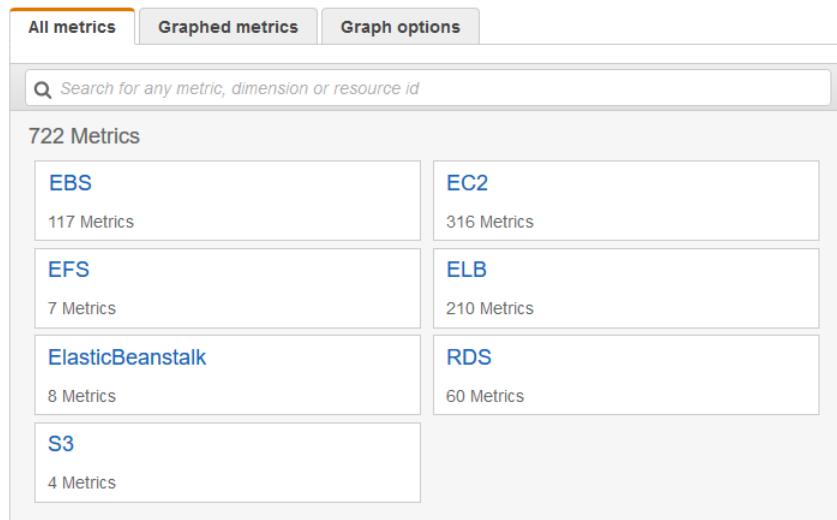
The following example shows you how to determine the maximum CPU utilization of a specific EC2 instance.

Requirements

- You must have the ID of the instance. You can get the instance ID using the Amazon EC2 console or the [describe-instances](#) command.
- By default, basic monitoring is enabled, but you can enable detailed monitoring. For more information, see [Enable or Disable Detailed Monitoring for Your Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

To display the average CPU utilization for a specific instance using the console

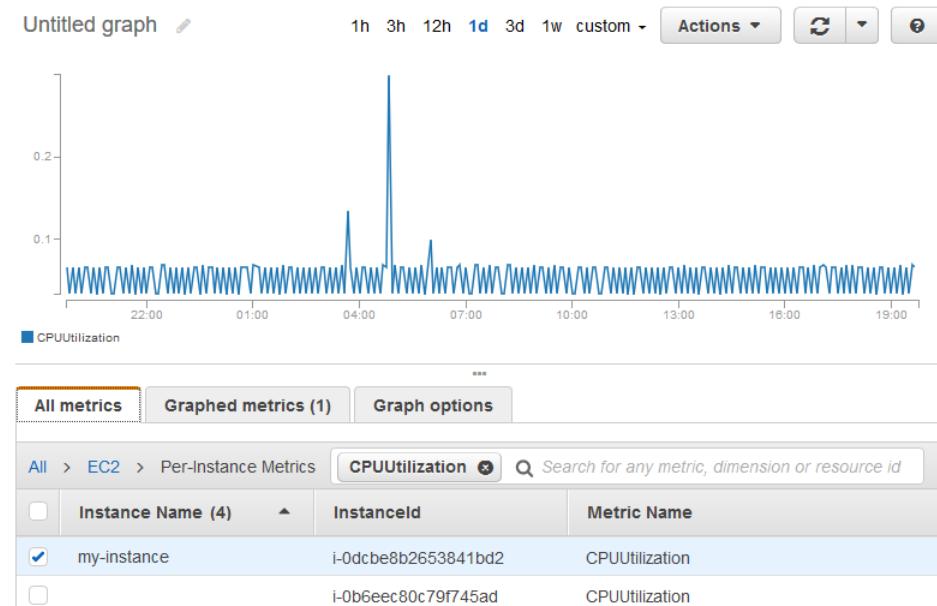
1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select the **EC2** metric namespace.



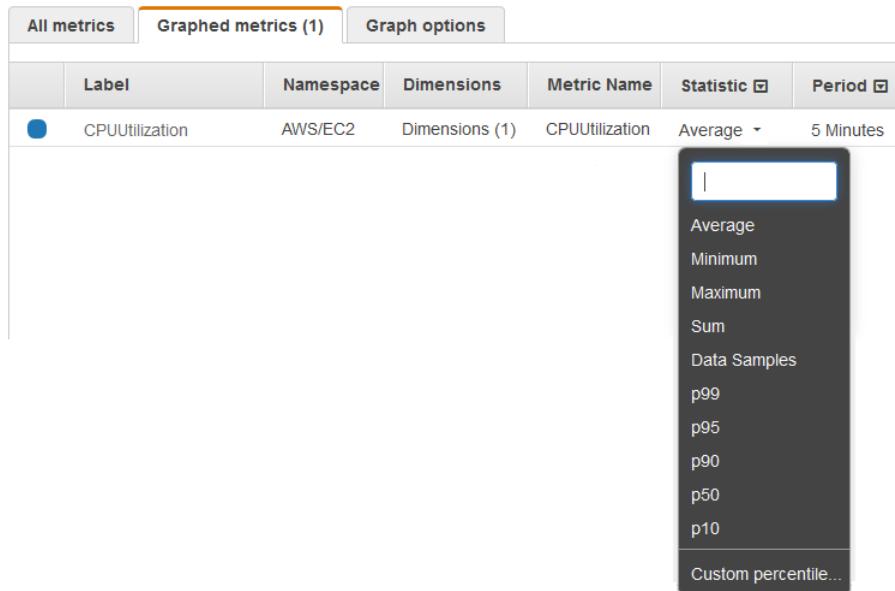
4. Select the **Per-Instance Metrics** dimension.

The screenshot shows the CloudWatch Metrics console. At the top, there are three tabs: "All metrics" (highlighted in orange), "Graphed metrics", and "Graph options". Below the tabs is a breadcrumb navigation bar: "All > EC2". To the right of the breadcrumb is a search bar with the placeholder text "Search for any metric, dimension or resource id". Underneath the search bar, it says "316 Metrics". There are two main sections: "By Auto Scaling Group" containing "124 Metrics" and "Per-Instance Metrics" containing "192 Metrics".

5. In the search field, enter **CPUutilization** and press Enter. Select the row for the specific instance, which displays a graph for the CPUUtilization metric for the instance. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



6. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p99.999**).



7. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value, and then choose a different value.

To get the CPU utilization per EC2 instance using the AWS CLI

Use the [get-metric-statistics](#) command as follows to get the CPUUtilization metric for the specified instance.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=i-1234567890abcdef0 --statistics Maximum \
--start-time 2016-10-18T23:18:00 --end-time 2016-10-19T23:18:00 --period 360
```

The returned statistics are 6-minute values for the requested 24-hour time interval. Each value represents the maximum CPU utilization percentage for the specified instance for a particular 6-minute time period. The data points aren't returned in chronological order. The following shows the beginning of the example output (the full output includes data points for every 6 minutes of the 24-hour period).

```
{
  "Datapoints": [
    {
      "Timestamp": "2016-10-19T00:18:00Z",
      "Maximum": 0.33000000000000002,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2016-10-19T03:18:00Z",
      "Maximum": 99.67000000000002,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2016-10-19T07:18:00Z",
      "Maximum": 0.3400000000000002,
      "Unit": "Percent"
    },
    ...
  ],
  "Label": "CPUUtilization"
```

}

Aggregating statistics across resources

You can aggregate the metrics for AWS resources across multiple resources. Metrics are completely separate between Regions, but you can use metric math to aggregate similar metrics across Regions. For more information, see [Using metric math \(p. 97\)](#).

For example, you can aggregate statistics for your EC2 instances that have detailed monitoring enabled. Instances that use basic monitoring aren't included. Therefore, you must enable detailed monitoring (at an additional charge), which provides data in 1-minute periods. For more information, see [Enable or Disable Detailed Monitoring for Your Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

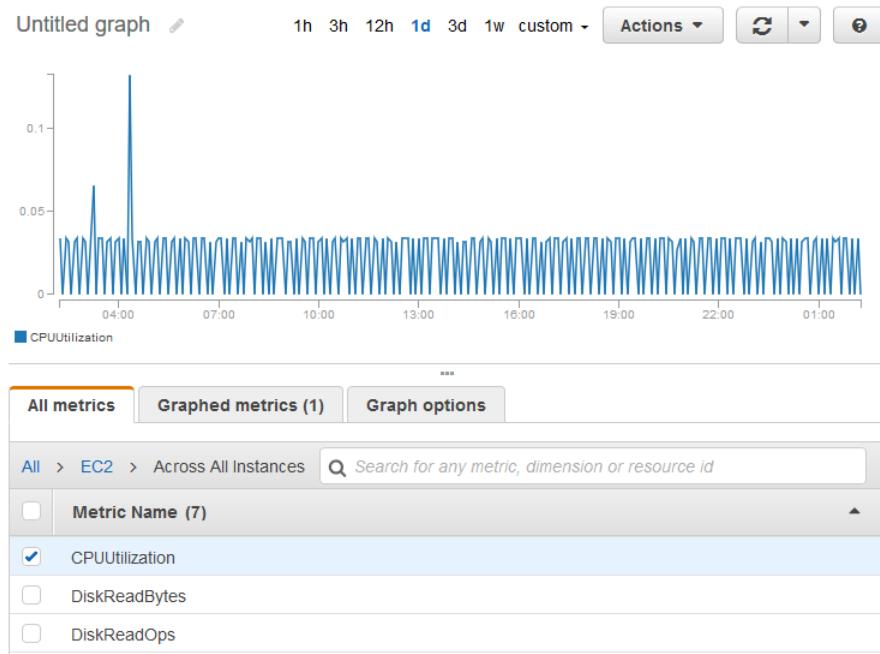
This example shows you how to get the average CPU usage for your EC2 instances. Because no dimension is specified, CloudWatch returns statistics for all dimensions in the AWS/EC2 namespace. To get statistics for other metrics, see [AWS services that publish CloudWatch metrics \(p. 746\)](#).

Important

This technique for retrieving all dimensions across an AWS namespace doesn't work for custom namespaces that you publish to CloudWatch. With custom namespaces, you must specify the complete set of dimensions that are associated with any given data point to retrieve statistics that include the data point.

To display average CPU utilization for your EC2 instances

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **EC2** namespace and choose **Across All Instances**.
4. Select the row that contains **CPUUtilization**, which displays a graph for the metric for all your EC2 instances. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).

6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get average CPU utilization across your EC2 instances using the AWS CLI

Use the [get-metric-statistics](#) command as follows:

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization --  
statistics "Average" "SampleCount" \  
--start-time 2016-10-11T23:18:00 --end-time 2016-10-12T23:18:00 --period 3600
```

The following is example output:

```
{  
    "Datapoints": [  
        {  
            "SampleCount": 238.0,  
            "Timestamp": "2016-10-12T07:18:00Z",  
            "Average": 0.038235294117647062,  
            "Unit": "Percent"  
        },  
        {  
            "SampleCount": 240.0,  
            "Timestamp": "2016-10-12T09:18:00Z",  
            "Average": 0.1667083333333332,  
            "Unit": "Percent"  
        },  
        {  
            "SampleCount": 238.0,  
            "Timestamp": "2016-10-11T23:18:00Z",  
            "Average": 0.041596638655462197,  
            "Unit": "Percent"  
        },  
        ...  
    ],  
    "Label": "CPUUtilization"  
}
```

Aggregating statistics by Auto Scaling group

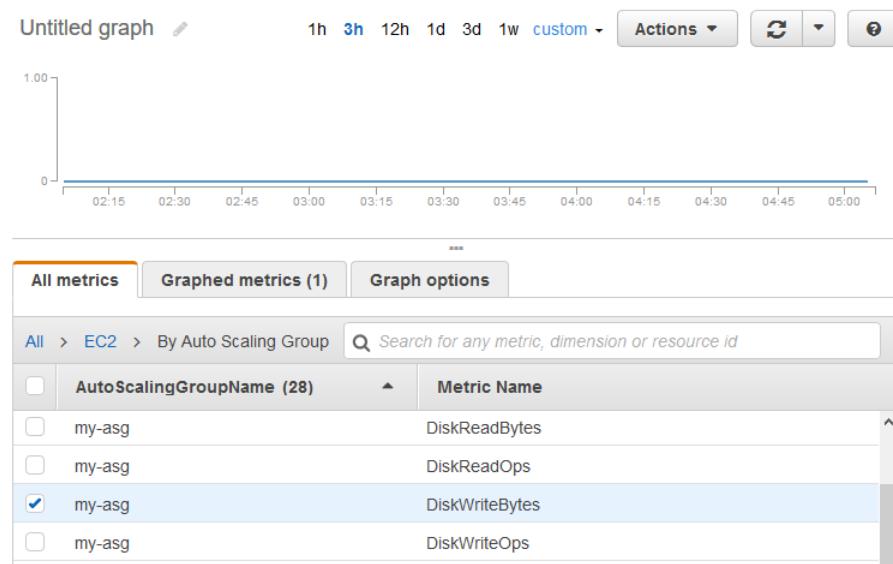
You can aggregate statistics for the EC2 instances in an Auto Scaling group. Metrics are completely separate between Regions, but you can use CloudWatch metric math to aggregate and transform metrics from multiple Regions. You can also use the cross-account dashboard to perform metric math on metrics from different accounts.

This example shows you how to get the total bytes written to disk for one Auto Scaling group. The total is computed for 1-minute periods for a 24-hour interval across all EC2 instances in the specified Auto Scaling group.

To display DiskWriteBytes for the instances in an Auto Scaling group using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **EC2** namespace and then choose **By Auto Scaling Group**.
4. Select the row for the **DiskWriteBytes** metric and the specific Auto Scaling group, which displays a graph for the metric for the instances in the Auto Scaling group. To change the name of the graph,

choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get DiskWriteBytes for the instances in an Auto Scaling group using the AWS CLI

Use the `get-metric-statistics` command as follows.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name DiskWriteBytes
--dimensions Name=AutoScalingGroupName,Value=my-asg --statistics "Sum" "SampleCount" \
--start-time 2016-10-16T23:18:00 --end-time 2016-10-18T23:18:00 --period 360
```

The following is example output.

```
{
  "Datapoints": [
    {
      "SampleCount": 18.0,
      "Timestamp": "2016-10-19T21:36:00Z",
      "Sum": 0.0,
      "Unit": "Bytes"
    },
    {
      "SampleCount": 5.0,
      "Timestamp": "2016-10-19T21:42:00Z",
      "Sum": 0.0,
      "Unit": "Bytes"
    }
  ],
  "Label": "DiskWriteBytes"
}
```

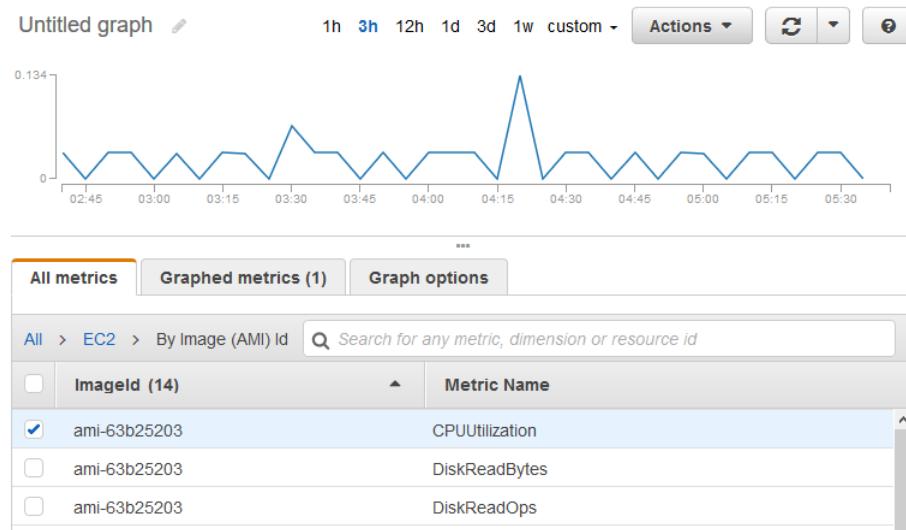
Aggregating statistics by Amazon Machine Image (AMI)

You can aggregate statistics for the EC2 instances that have detailed monitoring enabled. Instances that use basic monitoring aren't included. For more information, see [Enable or Disable Detailed Monitoring for Your Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

This example shows you how to determine average CPU utilization for all instances that use the specified AMI. The average is over 60-second time intervals for a one-day period.

To display the average CPU utilization by AMI using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **EC2** namespace and then choose **By Image (AMI) Id**.
4. Select the row for the **CPUUtilization** metric and the specific AMI, which displays a graph for the metric for the specified AMI. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get the average CPU utilization by AMI using the AWS CLI

Use the `get-metric-statistics` command as follows.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=ImageId,Value=ami-3c47a355 --statistics Average \
--start-time 2016-10-10T00:00:00 --end-time 2016-10-11T00:00:00 --period 3600
```

The operation returns statistics that are one-hour values for the one-day interval. Each value represents an average CPU utilization percentage for EC2 instances running the specified AMI. The following is example output.

```
{  
    "Datapoints": [  
        {  
            "Timestamp": "2016-10-10T07:00:00Z",  
            "Average": 0.04100000000000009,  
            "Unit": "Percent"  
        },  
        {  
            "Timestamp": "2016-10-10T14:00:00Z",  
            "Average": 0.079579831932773085,  
            "Unit": "Percent"  
        },  
        {  
            "Timestamp": "2016-10-10T06:00:00Z",  
            "Average": 0.03600000000000011,  
            "Unit": "Percent"  
        },  
        ...  
    ],  
    "Label": "CPUUtilization"  
}
```

Graphing metrics

Use the CloudWatch console to graph metric data generated by other AWS services. This makes it more efficient to see the metric activity on your services. The following procedures describe how to graph metrics in CloudWatch.

Contents

- [Graphing a metric \(p. 85\)](#)
- [Using dynamic labels \(p. 88\)](#)
- [Modifying the time range or time zone format for a graph \(p. 90\)](#)
- [Modifying the y-axis for a graph \(p. 93\)](#)
- [Creating an alarm from a metric on a graph \(p. 93\)](#)

Graphing a metric

You can select metrics and create graphs of the metric data using the CloudWatch console.

CloudWatch supports the following statistics on metrics: Average, Minimum, Maximum, Sum, and SampleCount. For more information, see [Statistics \(p. 5\)](#).

You can view your data at different levels of detail. For example, you can choose a one-minute view, which can be useful when troubleshooting. Or, choose a less detailed, one-hour view. That can be useful when viewing a broader time range (for example, 3 days) so that you can see trends over time. For more information, see [Periods \(p. 6\)](#).

Creating a graph

To graph a metric

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.

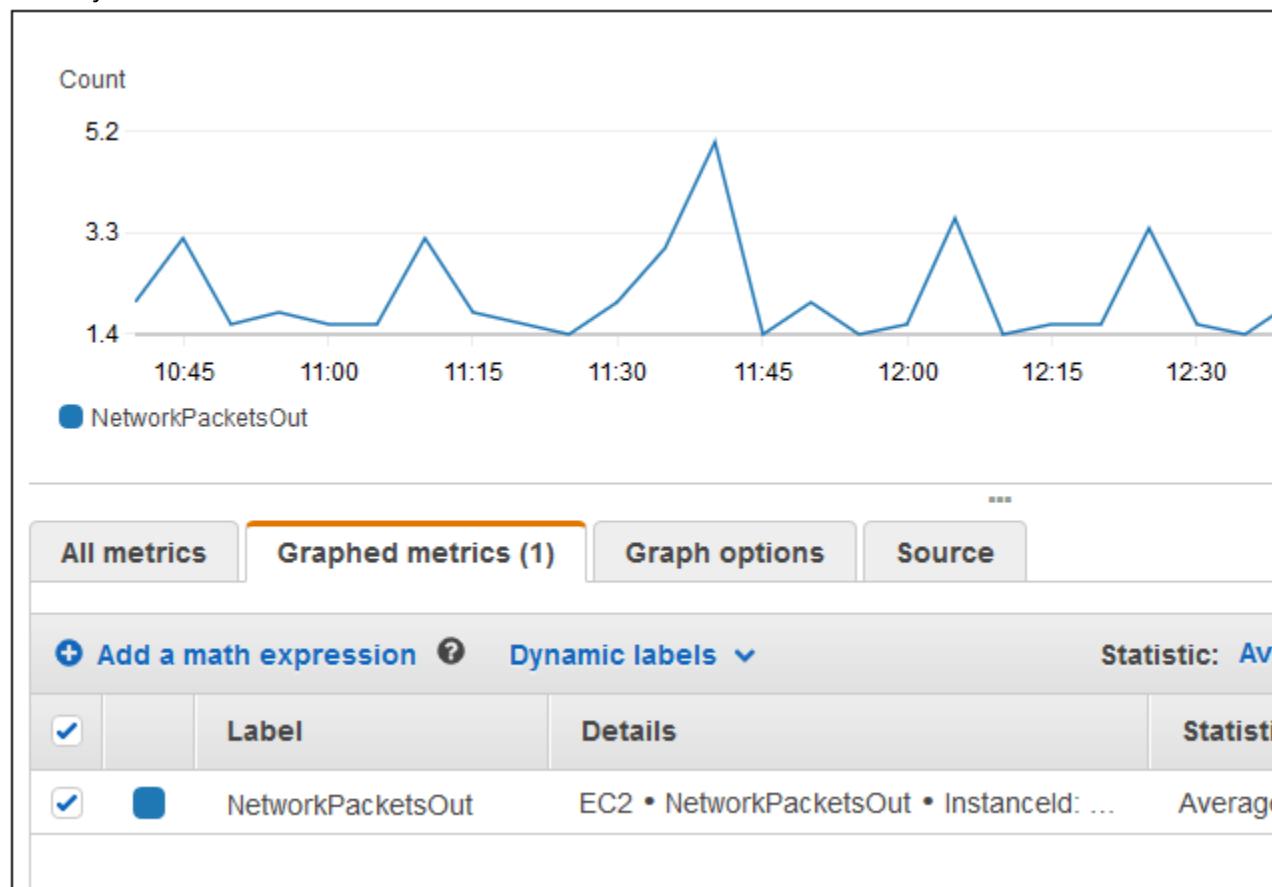
3. On the **All metrics** tab, enter a search term in the search field, such as a metric name or resource name, and press **Enter**.

For example, if you search for the `CPUUtilization` metric, you see the namespaces and dimensions with this metric.

4. Select one of the results for your search to view the metrics.
5. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
6. Choose **View graphed metrics**.
7. (Optional) To change the statistic used in the graph, choose the new statistic in the **Statistic** column next to the metric name.

For more information about CloudWatch statistics, see [CloudWatch statistics definitions \(p. 75\)](#).
For more information about the `pxx percentile` statistics, see [Percentiles \(p. 7\)](#).

8. (Optional) To change the type of graph, choose **Graph options**. You can then choose between a line graph, stacked area chart, bar chart, pie chart, or number.
9. (Optional) To add an anomaly detection band that shows expected values for the metric, choose the anomaly detection icon under **Actions** next to the metric.



CloudWatch uses up to two weeks of the metric's recent historical data to calculate a model for expected values. It then displays this range of expected values as a band on the graph. CloudWatch adds a new row under the metric to display the anomaly detection band math expression, labeled **ANOMALY_DETECTION_BAND**. If recent historical data exists, you immediately see a preview anomaly detection band, which is an approximation of the anomaly detection band generated by the model. It takes up to 15 minutes for the actual anomaly detection band to appear.

By default, CloudWatch creates the upper and lower bounds of the band of expected values with a default value of 2 for the band threshold. To change this number, change the value at the end of the formula under **Details** for the band.

- (Optional) Choose **Edit model** to change how the anomaly detection model is calculated. You can exclude past and future time periods from being used in the training for calculating the model. It is critical to exclude unusual events system as system outage, deployments, and holidays from the training data. You can also specify the time zone to use for the model for daylight saving time changes.

For more information, see [Using CloudWatch anomaly detection \(p. 289\)](#).

To hide the model from the graph, remove the checkmark from the line with the `ANOMALY_DETECTION_BAND` function or choose the X icon. To delete the model entirely, choose **Edit model**, **Delete model**.

10. (Optional) As you choose metrics to graph, specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric, and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics**, **Dynamic labels**.

By default, the dynamic values that you add to the label appear at the beginning of the label. You can then choose the **Label** value for the metric to edit the label. For more information, see [Using dynamic labels \(p. 88\)](#).

11. To view more information about the metric being graphed, pause the mouse over the legend.
12. Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.
 - b. For **Value**, enter the metric value where the horizontal annotation appears.
 - c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another **Value** field appears, and the area of the graph between the two values is filled.
 - d. For **Axis**, specify whether the numbers in **Value** refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose X in the **Actions** column.

13. To get a URL for your graph, choose **Actions**, **Share**. Copy the URL to save or share.
14. To add your graph to a dashboard, choose **Actions**, **Add to dashboard**.

Updating a graph

To update your graph

1. To change the name of the graph, choose the pencil icon.

2. To change the time range, select one of the predefined values or choose **custom**. For more information, see [Modifying the time range or time zone format for a graph \(p. 90\)](#).
3. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95 . 45**).
4. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.
5. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.
 - b. For **Value**, enter the metric value where the horizontal annotation appears.
 - c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose **Above** or **Below** for the corresponding area to be filled. If you specify **Between**, another **Value** field appears, and the area of the graph between the two values is filled.
 - d. For **Axis**, specify whether the numbers in **Value** refer to the metric associated with the left y-axis or the right y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

6. To change the refresh interval, choose **Refresh options** and then select **Auto refresh** or choose **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Duplicating a metric

To duplicate a metric

1. Choose the **Graphed metrics** tab.
2. For **Actions**, choose the **Duplicate** icon.



3. Update the duplicate metric as needed.

Using dynamic labels

You can use dynamic labels with your graphs. Dynamic labels add a dynamically updated value to the label for the selected metric. You can add a wide range of values to the labels, as shown in the following tables.

The dynamic value shown in the label is derived from the time range currently shown on the graph. The dynamic part of the label automatically updates when either the dashboard or the graph is refreshed.

If you use a dynamic label with a search expression, the dynamic label applies to every metric returned by the search.

You can use the CloudWatch console to add a dynamic value to a label, edit the label, change the position of the dynamic value within the label column, and make other customizations.

Dynamic labels

Within a dynamic label, you can use the following values relating to properties of the metric:

| Dynamic label live value | Description |
|---|--|
| <code> \${AVG}</code> | The average of the values in the time range currently shown in the graph. |
| <code> \${DATAPPOINT_COUNT}</code> | The number of data points in the time range that is currently shown in the graph. |
| <code> \${FIRST}</code> | The oldest of the metric values in the time range that is currently shown in the graph. |
| <code> \${FIRST_LAST_RANGE}</code> | The difference between the metric values of the oldest and newest data points that are currently shown in the graph. |
| <code> \${FIRST_LAST_TIME_RANGE}</code> | The absolute time range between the oldest and newest data points that are currently shown in the graph. |
| <code> \${FIRST_TIME}</code> | The timestamp of the oldest data point in the time range that is currently shown in the graph. |
| <code> \${FIRST_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the oldest data point in the time range that is currently shown in the graph. |
| <code> \${LABEL}</code> | The representation of the default label for a metric. |
| <code> \${LAST}</code> | The most recent of the metric values in the time range that is currently shown in the graph. |
| <code> \${LAST_TIME}</code> | The timestamp of the newest data point in the time range that is currently shown in the graph. |
| <code> \${LAST_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the newest data point in the time range that is currently shown in the graph. |
| <code> \${MAX}</code> | The maximum of the values in the time range currently shown in the graph. |
| <code> \${MAX_TIME}</code> | The timestamp of the data point that has the highest metric value, of the data points that are currently shown in the graph. |
| <code> \${MAX_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the data point with the highest value, of those data points that are currently shown in the graph. |
| <code> \${MIN}</code> | The minimum of the values in the time range currently shown in the graph. |

| Dynamic label live value | Description |
|--|---|
| <code> \${MIN_MAX_RANGE}</code> | The difference in metric values between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph. |
| <code> \${MIN_MAX_TIME_RANGE}</code> | The absolute time range between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph. |
| <code> \${MIN_TIME}</code> | The timestamp of the data point that has the lowest metric value, of the data points that are currently shown in the graph. |
| <code> \${MIN_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the data point with the lowest value, of those data points that are currently shown in the graph. |
| <code> \${PROP('AccountId')}</code> | The AWS account ID of the metric. |
| <code> \${PROP('Dim.<i>dimension_name</i>'')}</code> | The value of the specified dimension. |
| <code> \${PROP('MetricName')}</code> | The name of the metric. |
| <code> \${PROP('Namespace')}</code> | The namespace of the metric. |
| <code> \${PROP('Period')}</code> | The period of the metric, in seconds. |
| <code> \${PROP('Region')}</code> | The AWS Region where the metric is published. |
| <code> \${PROP('Stat')}</code> | The metric statistic that is being graphed. |
| <code> \${SUM}</code> | The sum of the values in the time range currently shown in the graph. |

For example, suppose you have a search expression `SEARCH(' {AWS/Lambda, FunctionName} Errors ', 'Sum', 300)`, which finds the Errors for each of your Lambda functions. If you set the label to be `[max: ${MAX} Errors for Function Name ${LABEL}]`, the label for each metric is **[max: *number* Errors for Function Name *Name*]**.

You can add up to five dynamic values to a label. You can use the `${LABEL}` placeholder only once within each label.

Modifying the time range or time zone format for a graph

This section describes how you can modify the date, time, and time zone format on a CloudWatch metrics graph. It also describes how you can zoom in on a graph to apply a specific time range. For information about creating a graph, see [Graphing a metric](#).

Setting a relative time range

New interface

To specify a relative time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Relative** tab in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, **Months**.
4. After you specify a time range, choose **Apply**.

Original interface

To specify a relative time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Relative** in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, or **Months**.

Setting an absolute time range

New interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Absolute** tab in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Original interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Absolute** in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Setting the time zone format

New interface

To specify the time zone for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local time zone**.
4. After you make your changes, choose **Apply**.

Original interface

To specify the time zone for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local timezone**.

Zooming in on a graph

New interface

To zoom in on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose and drag on the area of the graph that you want to zoom in on, and then release the drag. You can apply a time range to the graph.
4. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Original interface

To zoom in on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.

3. Choose and drag on the area of the graph that you want to zoom in on, and then release the drag. You can apply a time range to the graph.
4. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Modifying the y-axis for a graph

You can set custom bounds for the y-axis on a graph to help you see the data better. For example, you can change the bounds on a CPUUtilization graph to 100 percent so that it's easy to see whether the CPU is low (the plotted line is near the bottom of the graph) or high (the plotted line is near the top of the graph).

You can switch between two different y-axes for your graph. This is useful if the graph contains metrics that have different units or that differ greatly in their range of values.

To modify the y-axis on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).
4. The **All metrics** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. On the **Graph options** tab, specify the **Min** and **Max** values for **Left Y Axis**. The value of **Min** can't be greater than the value of **Max**.

The screenshot shows the 'Graph options' tab selected. Under 'Left Y Axis', there are 'Limits' fields for 'Min' (0) and 'Max' (100). Under 'Right Y Axis', there are 'Limits' fields for 'Min' (Auto) and 'Max' (Auto).

6. To create a second y-axis, specify the **Min** and **Max** values for **Right Y Axis**.
7. To switch between the two y-axes, choose the **Graphed metrics** tab. For **Y Axis**, choose **Left Y Axis** or **Right Y Axis**.

The screenshot shows the 'Graphed metrics' tab selected. At the bottom right, there is a 'Y Axis' dropdown menu with 'Left Y Axis' and 'Right Y Axis' options. A cursor is hovering over the 'Right Y Axis' option.

Creating an alarm from a metric on a graph

You can graph a metric and then create an alarm from the metric on the graph, which has the benefit of populating many of the alarm fields for you.

To create an alarm from a metric on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).
4. The **All metrics** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. To create an alarm for the metric, choose the **Graphed metrics** tab. For **Actions**, choose the alarm icon.



6. Under **Conditions**, choose **Static** or **Anomaly detection** to specify whether to use a static threshold or anomaly detection model for the alarm.

Depending on your choice, enter the rest of the data for the alarm conditions.

7. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the **ALARM** state to trigger the alarm. If the two values here match, you create an alarm that goes to **ALARM** state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm \(p. 131\)](#).

8. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data \(p. 132\)](#).
9. Choose **Next**.
10. Under **Notification**, select an SNS topic to notify when the alarm is in **ALARM** state, **OK** state, or **INSUFFICIENT_DATA** state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

11. To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.
12. When finished, choose **Next**.
13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Publishing custom metrics

You can publish your own metrics to CloudWatch using the AWS CLI or an API. You can view statistical graphs of your published metrics with the AWS Management Console.

CloudWatch stores data about a metric as a series of data points. Each data point has an associated time stamp. You can even publish an aggregated set of data points called a *statistic set*.

Topics

- [High-resolution metrics \(p. 95\)](#)
- [Using dimensions \(p. 95\)](#)
- [Publishing single data points \(p. 96\)](#)
- [Publishing statistic sets \(p. 97\)](#)
- [Publishing the value zero \(p. 97\)](#)

High-resolution metrics

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling `PutMetricData` more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Using dimensions

In custom metrics, the `--dimensions` parameter is common. A dimension further clarifies what the metric is and what data it stores. You can have up to 10 dimensions in one metric, and each dimension is defined by a name and value pair.

How you specify a dimension is different when you use different commands. With `put-metric-data`, you specify each dimension as `Name=MyName,Value=MyValue`, and with `get-metric-statistics` or `put-metric-alarm` you use the format `Name=MyName,Value=MyValue`. For example, the following command publishes a `Buffers` metric with two dimensions named `InstanceId` and `InstanceType`.

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit Bytes
--value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

This command retrieves statistics for that same metric. Separate the Name and Value parts of a single dimension with commas, but if you have multiple dimensions, use a space between one dimension and the next.

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --
dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-time
2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period 60
```

If a single metric includes multiple dimensions, you must specify a value for every defined dimension when you use [get-metric-statistics](#). For example, the Amazon S3 metric `BucketSizeBytes` includes the dimensions `BucketName` and `StorageType`, so you must specify both dimensions with [get-metric-statistics](#).

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --start-time 2017-01-23T14:23:00Z --end-time 2017-01-26T19:30:00Z --period 3600 --namespace AWS/S3 --statistics Maximum --dimensions Name=BucketName,Value=MyBucketName Name=StorageType,Value=StandardStorage --output table
```

To see what dimensions are defined for a metric, use the [list-metrics](#) command.

Publishing single data points

To publish a single data point for a new or existing metric, use the [put-metric-data](#) command with one value and time stamp. For example, the following actions each publish one data point.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value 5 --timestamp 2016-10-20T12:00:02.000Z
```

If you call this command with a new metric name, CloudWatch creates a metric for you. Otherwise, CloudWatch associates your data with the existing metric that you specified.

Note

When you create a metric, it can take up to 2 minutes before you can retrieve statistics for the new metric using the [get-metric-statistics](#) command. However, it can take up to 15 minutes before the new metric appears in the list of metrics retrieved using the [list-metrics](#) command.

Although you can publish data points with time stamps as granular as one-thousandth of a second, CloudWatch aggregates the data to a minimum granularity of 1 second. CloudWatch records the average (sum of all items divided by number of items) of the values received for each period, as well as the number of samples, maximum value, and minimum value for the same time period. For example, the `PageViewCount` metric from the previous examples contains three data points with time stamps just seconds apart. If you have your period set to 1 minute, CloudWatch aggregates the three data points because they all have time stamps within a 1-minute period.

You can use the [get-metric-statistics](#) command to retrieve statistics based on the data points that you published.

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount \
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" \
--start-time 2016-10-20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

The following is example output.

```
{  
    "Datapoints": [  
        {  
            "SampleCount": 3.0,  
            "Timestamp": "2016-10-20T12:00:00Z",  
            "Average": 3.6666666666666665,  
            "Maximum": 5.0,  
            "Minimum": 2.0,  
        }  
    ]  
}
```

```
        "Sum": 11.0,  
        "Unit": "None"  
    }  
],  
"Label": "PageViewCount"  
}
```

Publishing statistic sets

You can aggregate your data before you publish to CloudWatch. When you have multiple data points per minute, aggregating data minimizes the number of calls to **put-metric-data**. For example, instead of calling **put-metric-data** multiple times for three data points that are within 3 seconds of each other, you can aggregate the data into a statistic set that you publish with one call, using the **--statistic-values** parameter.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService  
--statistic-values Sum=11,Minimum=2,Maximum=5,SampleCount=3 --  
timestamp 2016-10-14T12:00:00.000Z
```

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can't retrieve percentile statistics for this data unless one of the following conditions is true:

- The **SampleCount** of the statistic set is 1
- The **Minimum** and the **Maximum** of the statistic set are equal

Publishing the value zero

When your data is more sporadic and you have periods that have no associated data, you can choose to publish the value zero (0) for that period or no value at all. If you use periodic calls to **PutMetricData** to monitor the health of your application, you might want to publish zero instead of no value. For example, you can set a CloudWatch alarm to notify you if your application fails to publish metrics every five minutes. You want such an application to publish zeros for periods with no associated data.

You might also publish zeros if you want to track the total number of data points or if you want statistics such as minimum and average to include data points with the value 0.

Using metric math

Metric math enables you to query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series on the CloudWatch console and add them to dashboards. Using AWS Lambda metrics as an example, you could divide the **Errors** metric by the **Invocations** metric to get an error rate. Then add the resulting time series to a graph on your CloudWatch dashboard.

You can also perform metric math programmatically, using the **GetMetricData** API operation. For more information, see [GetMetricData](#).

Adding a math expression to a CloudWatch graph

You can add a math expression to a graph on your CloudWatch dashboard. Each graph is limited to using a maximum of 500 metrics and expressions, so you can add a math expression only if the graph has 499 or fewer metrics. This applies even if not all the metrics are displayed on the graph.

To add a math expression to a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Create or edit a graph. There needs to be at least one metric in the graph.
3. Choose **Graphed metrics**.
4. Choose **Math expression, Start with empty expression**. A new line appears for the expression.
5. In the new line, under the **Details** column, enter the math expression. The tables in the Metric Math Syntax and Functions section list the functions that you can use in the expression.

To use a metric or the result of another expression as part of the formula for this expression, use the value shown in the **Id** column: for example, **m1+m2** or **e1-MIN(e1)**.

You can change the value of **Id**. It can include numbers, letters, an underscore, and must start with a lowercase letter. Changing the value of **Id** to a more meaningful name can also make a graph easier to understand; for example, changing **m1** and **m2** to **errors** and **requests**.

Tip

Choose the down arrow next to **Math Expression** to see a list of supported functions, which you can use when creating your expression.

6. For the **Label** column of the expression, enter a name that describes what the expression is calculating.

If the result of an expression is an array of time series, each of those time series is displayed on the graph with a separate line, with different colors. Immediately under the graph is a legend for each line in the graph. For a single expression that produces multiple time series, the legend captions for those time series are in the format **Expression-Label Metric-Label**. For example, if the graph includes a metric with a label of **Errors** and an expression **FILL(METRICS(), 0)** that has a label of **Filled With 0**, one line in the legend would be **Filled With 0: Errors**. To have the legend show only the original metric labels, set **Expression-Label** to be empty.

When one expression produces an array of time series on the graph, you can't change the colors used for each of those time series.

7. After you have added the desired expressions, you can simplify the graph by hiding some of the original metrics. To hide a metric or expression, clear the check box to the left of the **Id** field.

Metric math syntax and functions

The following sections explain the functions available for metric math. All functions must be written in uppercase letters (such as **AVG**), and the **Id** field for all metrics and math expressions must start with a lowercase letter.

The final result of any math expression must be a single time series or an array of time series. Some functions produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value in the time series.

Data type abbreviations

Some functions are valid for only certain types of data. The abbreviations in the following list are used in the tables of functions to represent the types of data supported for each function:

- **S** represents a scalar number, such as 2, -5, or 50.25
- **TS** is a time series (a series of values for a single CloudWatch metric over time): for example, the **CPUUtilization** metric for instance **i-1234567890abcdef0** over the last 3 days

- **TS[]** is an array of time series, such as the time series for multiple metrics

The METRICS() function

The **METRICS()** function returns all the metrics in the request. Math expressions aren't included.

You can use **METRICS()** within a larger expression that produces a single time series or an array of time series. For example, the expression **SUM(METRICS())** returns a time series (TS) that is the sum of the values of all the graphed metrics. **METRICS()/100** returns an array of time series, each of which is a time series showing each data point of one of the metrics divided by 100.

You can use the **METRICS()** function with a string to return only the graphed metrics that contain that string in their **Id** field. For example, the expression **SUM(METRICS("errors"))** returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their **Id** field. You can also use **SUM([METRICS("4xx"), METRICS("5xx")])** to match multiple strings.

Basic arithmetic functions

The following table lists the basic arithmetic functions that are supported. Missing values in a time series are treated as 0. If the value of a data point causes a function to attempt to divide by zero, the data point is dropped.

| Operation | Arguments | Examples |
|---------------------------------|--|--|
| Arithmetic operators: + - * / ^ | S, S
S, TS
TS, TS
S, TS[]
TS, TS[] | PERIOD(m1)/60
5 * m1
m1 - m2
SUM(100/[m1, m2])
AVG(METRICS())
METRICS()*100 |
| Unary subtraction - | S
TS
TS[] | -5*m1
-m1
SUM(-[m1, m2]) |

Comparison and logical operators

You can use comparison and logical operators with either a pair of time series or a pair of single scalar values. When you use a comparison operator with a pair of time series, the operators return a time series where each data point is either 0 (false) or 1 (true). If you use a comparison operator on a pair of scalar values, a single scalar value is returned, either 0 or 1.

When comparison operators are used between two time series, and only one of the time series has a value for a particular time stamp, the function treats the missing value in the other time series as **0**.

You can use logical operators in conjunction with comparison operators, to create more complex functions.

The following table lists the operators that are supported.

| Type of operator | Supported operators |
|----------------------|--|
| Comparison operators | <code>==</code>
<code>!=</code>
<code><=</code>
<code>>=</code>
<code><</code>
<code>></code> |
| Logical operators | <code>AND</code> or <code>&&</code>
<code>OR</code> or <code> </code> |

To see how these operators are used, suppose we have two time series: **metric1** has values of [30, 20, 0, 0] and **metric2** has values of [20, -, 20, -] where – indicates that there is no value for that timestamp.

| Expression | Output |
|--|-------------------------|
| <code>(metric1 < metric2)</code> | <code>0, 0, 1, 0</code> |
| <code>(metric1 >= 30)</code> | <code>1, 0, 0, 0</code> |
| <code>(metric1 > 15 AND metric2 > 15)</code> | <code>1, 0, 0, 0</code> |

Functions supported for metric math

The following table describes the functions that you can use in math expressions. Enter all functions in uppercase letters.

The final result of any math expression must be a single time series or an array of time series. Some functions in tables in the following sections produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value of that data point.

In the following table, every example in the **Examples** column is an expression that results in a single time series or an array of time series. These examples show how functions that return scalar numbers can be used as part of a valid expression that produces a single time series.

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|------------|------------|--------------|--|---|------------------------------|
| ABS | TS
TS[] | TS
TS[] | Returns the absolute value of each data point. | <code>ABS(m1-m2)</code>
<code>MIN(ABS([m1, m2]))</code>
<code>ABS(METRICS())</code> | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|---------------------------------|-----------|--------------|---|--|------------------------------|
| ANOMALY_DETECTION_BAND() | TS, S | | Returns an anomaly detection band for the specified metric. The band consists of two time series, one representing the upper limit of the "normal" expected value of the metric, and the other representing the lower limit. The function can take two arguments. The first is the ID of the metric to create the band for. The second argument is the number of standard deviations to use for the band. If you don't specify this argument, the default of 2 is used. For more information, see Using CloudWatch anomaly detection (p. 289) . | ANOMALY_DETECTION_BAND(m1)
ANOMALY_DETECTION_BAND(m1,4) | |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-------------|------------|--------------|--|--|------------------------------|
| AVG | TS
TS[] | S
TS | The AVG of a single time series returns a scalar representing the average of all the data points in the metric. The AVG of an array of time series returns a single time series. Missing values are treated as 0.

Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | SUM([m1,m2])/AVG(m2)

AVG(METRICS()) | ✓ |
| CEIL | TS
TS[] | TS
TS[] | Returns the ceiling of each metric. The ceiling is the smallest integer greater than or equal to each value. | CEIL(m1)

CEIL(METRICS())

SUM(CEIL(METRICS())) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|------------------------|------------|--------------|--|---|------------------------------|
| DATAPOINT_COUNT | TS[] | S
TS | Returns a count of the data points that reported values. This is useful for calculating averages of sparse metrics.

Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | SUM(m1) / DATAPOINT_COUNT(m1)

DATAPOINT_COUNT(METRICS()) | ✓ |
| DIFF | TS
TS[] | TS
TS[] | Returns the difference between each value in the time series and the preceding value from that time series. | DIFF(m1) | ✓ |
| DIFF_TIME | TS
TS[] | TS
TS[] | Returns the difference in seconds between the timestamp of each value in the time series and the timestamp of the preceding value from that time series. | DIFF_TIME(METRICS()) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-------------|---|--------------|---|--|------------------------------|
| FILL | TS, [S REPEAT LINEAR]

TS[], [TS S REPEAT LINEAR] | TS
TS[] | <p>Fills the missing values of a time series. There are several options for the values to use as the filler for missing values:</p> <ul style="list-style-type: none"> You can specify a value to use as the filler value. You can specify a metric to use as the filler value. You can use the REPEAT keyword to fill missing values with the most recent actual value of the metric before the missing value. You can use the LINEAR keyword to fill the missing values with values that create a linear interpolation between the values at the beginning and the end of the gap. <p>Note
When you use this function in an alarm, you can encounter an issue if your metrics are being published with a slight delay, and the most recent minute never has data. In this case, FILL replaces that missing data point with the requested value. That causes the latest data point for the metric to always be the FILL value, which can result in the alarm being stuck in either OK state or ALARM</p> | FILL(m1,10)
FILL(METRICS(), 0)
FILL(METRICS(), m1)
FILL(m1, MIN(m1))
FILL(m1, REPEAT)
FILL(METRICS(), LINEAR) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-----------------------------|----------------------------|--------------|---|--|------------------------------|
| | | | state. You can work around this by using a M out of N alarm. For more information, see Evaluating an alarm (p. 131) . | | |
| FIRST
LAST | TS[] | TS | Returns the first or last time series from an array of time series. This is useful when used with the SORT function. It can also be used to get the high and low thresholds from the ANOMALY_DETECTION_BAND function. | IF(FIRST(SORT(METRICS()), AVG, DESC))>100, 1, 0)
Looks at the top metric from an array, which is sorted by AVG. It then returns a 1 or 0 for each data point, depending on whether that data point is more than 100.

LAST(ANOMALY_DETECTION_BAND(m1)) returns the lower bound of the anomaly prediction band. | ✓ |
| FLOOR | TS
TS[] | TS
TS[] | Returns the floor of each metric. The floor is the largest integer less than or equal to each value. | FLOOR(m1)
FLOOR(METRICS()) | ✓ |
| IF | IF expression | TS | Use IF along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series. For more information, see Using IF expressions (p. 116) . | For examples, see Using IF expressions (p. 116) . | ✓ |
| INSIGHT_RULE_METRIC | INSIGHT_RULE_METRIC | Metric | METRIC(ruleName, INSIGHT_RULE_METRIC to extract statistics from a rule in Contributor Insights. For more information, see Graphing metrics generated by rules (p. 302) . | | |
| LOG | TS
TS[] | TS
TS[] | The LOG of a time series returns the natural logarithm value of each value in the time series. | LOG(METRICS()) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|---------------------|------------|--------------|---|--|------------------------------|
| LOG10 | TS
TS[] | TS
TS[] | The LOG10 of a time series returns the base-10 logarithm value of each value in the time series. | LOG10(m1) | ✓ |
| MAX | TS
TS[] | S
TS | The MAX of a single time series returns a scalar representing the maximum value of all data points in the metric. The MAX value of an array of time series returns a single time series.

Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | MAX(m1)/m1

MAX(METRICS()) | ✓ |
| METRIC_COUNT | TS[] | S | Returns the number of metrics in the time series array. | m1/METRIC_COUNT(METRICS()) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|----------------|-------------|--------------|---|--|------------------------------|
| METRICS | null string | TS[] | <p>The METRICS() function returns all CloudWatch metrics in the request. Math expressions aren't included.</p> <p>You can use METRICS() within a larger expression that produces a single time series or an array of time series.</p> <p>You can use the METRICS() function with a string to return only the graphed metrics that contain that string in their Id field. For example, the expression SUM(METRICS("errors")) returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their Id field. You can also use SUM([METRICS("4xx"), METRICS("5xx")]) to match multiple strings.</p> | AVG(METRICS())
SUM(METRICS("errors")) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|------------|------------|--------------|---|--|------------------------------|
| MIN | TS
TS[] | S
TS | The MIN of a single time series returns a scalar representing the minimum value of all data points in the metric. The MIN of an array of time series returns a single time series.

Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | m1-MIN(m1)

MIN(METRICS()) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|--|----------|--------------|---|--|------------------------------|
| MINUTE
HOUR
DAY
DATE
MONTH
YEAR
EPOCH | TS | TS | <p>These functions take the period and range of the time series and return a new non-sparse time series where each value is based on its timestamp.</p> <ul style="list-style-type: none"> • MINUTE returns a non-sparse time series of integers between 0 and 59 that represent the UTC minute of each timestamp in the original time series. • HOUR returns a non-sparse time series of integers between 0 and 23 that represent the UTC hour of each timestamp in the original time series. • DAY returns a non-sparse time series of integers between 1 and 7 that represent the UTC day of the week of each timestamp in the original time series. 1 represents Monday and 7 represents Sunday. • DATE returns a non-sparse time series of integers between 1 and 31 that represent the UTC day of the month of each timestamp in the original time series. • MONTH returns a non-sparse time series of integers between 1 and 12 that represent the UTC month of each timestamp in the original time series. 1 represents January and 12 represents December. • YEAR returns a non-sparse time series of integers that represent the UTC year of each | MINUTE(m1)

IF(DAY(m1)<6,m1)
returns metrics only from weekdays, Monday to Friday.

IF(MONTH(m1) == 4,m1)
returns only metrics published in April. | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|---------------|------------|--------------|---|---|------------------------------|
| | | | <p>timestamp in the original time series.</p> <ul style="list-style-type: none"> • EPOCH returns a non-sparse time series of integers that represent the UTC time in seconds since the Epoch of each timestamp in the original time series. The Epoch is January 1, 1970. | | |
| PERIOD | TS | S | Returns the period of the metric in seconds. Valid input is metrics, not the results of other expressions. | m1/PERIOD(m1) | ✓ |
| RATE | TS
TS[] | TS
TS[] | Returns the rate of change of the metric per second. This is calculated as the difference between the latest data point value and the previous data point value, divided by the time difference in seconds between the two values. | RATE(m1)
RATE(METRICS()) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|---------------------|-----------|--------------|---|--------------------------------|------------------------------|
| REMOVE_EMPTY | TS[] | TS[] | <p>Removes any time series that have no data points from an array of time series. The result is an array of time series where each time series contains at least one data point.</p> <p>Note
 We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.</p> | REMOVE_EMPTY(METRICS()) | |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|--------------------|-------------------|----------------|---|-----------------------------|------------------------------|
| RUNNING_SUM | TS
TS[] | TS
TS[] | Returns a time series with the running sum of the values in the original time series.

Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | RUNNING_SUM([m1,m2]) | ✓ |
| SEARCH | Search expression | One or more TS | Returns one or more time series that match a search criteria that you specify. The SEARCH function enables you to add multiple related time series to a graph with one expression. The graph is dynamically updated to include new metrics that are added later and match the search criteria. For more information, see Using search expressions in graphs (p. 119) .

You can't create an alarm based on a SEARCH expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series. | | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|----------------------|---------------------------|--------------|--|--|------------------------------|
| SERVICE_QUOTA | TS that is a usage metric | TS | Returns the service quota for the given usage metric. You can use this to visualize how your current usage compares to the quota, and to set alarms that warn you when you approach the quota. For more information, see AWS usage metrics (p. 832) . | | |
| SLICE | (TS[], S, S) or (TS[], S) | TS[]
TS | <p>Retrieves part of an array of time series. This is especially useful when combined with SORT. For example, you can exclude the top result from an array of time series.</p> <p>You can use two scalar arguments to define the set of time series that you want returned. The two scalars define the start (inclusive) and end (exclusive) of the array to return. The array is zero-indexed, so the first time series in the array is time series 0. Alternatively, you can specify just one value, and CloudWatch returns all time series starting with that value.</p> | <p>SLICE(SORT(METRICS()), SUM, DESC), 0, 10) returns the 10 metrics from the array of metrics in the request that have the highest SUM value.</p> <p>SLICE(SORT(METRICS()), AVG, ASC), 5 sorts the array of metrics by the AVG statistic, then returns all the time series except for the 5 with the lowest AVG.</p> | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-------------|---|--------------|---|---|------------------------------|
| SORT | (TS[], FUNCTION, SORT_ORDER)

(TS[], FUNCTION, SORT_ORDER, S) | TS[] | <p>Sorts an array of time series according to the function you specify. The function you use can be AVG, MIN, MAX, or SUM. The sort order can be either ASC for ascending (lowest values first) or DESC to sort the higher values first. You can optionally specify a number after the sort order which acts as a limit. For example, specifying a limit of 5 returns only the top 5 time series from the sort.</p> <p>When this math function is displayed on a graph, the labels for each metric in the graph are also sorted and numbered.</p> | <p>SORT(METRICS(), AVG, DESC, 10) calculates the average value of each time series, sorts the time series with the highest values at the beginning of the sort, and returns only the 10 time series with the highest averages.</p> <p>SORT(METRICS(), MAX, ASC) sorts the array of metrics by the MAX statistic, then returns all of them in ascending order.</p> | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|---------------|------------|--------------|--|--|------------------------------|
| STDDEV | TS
TS[] | S
TS | <p>The STDDEV of a single time series returns a scalar representing the standard deviation of all data points in the metric. The STDDEV of an array of time series returns a single time series.</p> <p>Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.</p> | m1/STDDEV(m1)
STDDEV(METRICS()) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|--------------------|------------|--------------|---|--|------------------------------|
| SUM | TS
TS[] | S
TS | The SUM of a single time series returns a scalar representing the sum of the values of all data points in the metric. The SUM of an array of time series returns a single time series.

Note
We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | SUM(METRICS())/SUM(m1)

SUM([m1,m2])

SUM(METRICS("errors"))/SUM(METRICS("requests"))*100 | ✓ |
| TIME_SERIES | S | TS | Returns a non-sparse time series where every value is set to a scalar argument. | TIME_SERIES(MAX(m1))

TIME_SERIES(5*AVG(m1))

TIME_SERIES(10) | ✓ |

*Using a function that returns only a scalar number is not valid, as all final results of expressions must be a single time series or an array of time series. Instead, use these functions as part of a larger expression that returns a time series.

Using IF expressions

Use **IF** along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series.

IF uses the following arguments:

```
IF(condition, trueValue, falseValue)
```

The condition evaluates to FALSE if the value of the condition data point is 0, and to TRUE if the value of the condition is any other value, whether that value is positive or negative. If the condition is a time series, it is evaluated separately for every timestamp.

The following lists the valid syntaxes. For each of these syntaxes, the output is a single time series.

- **IF(TS *Comparison Operator* S, S | TS, S / TS)**
- **IF(TS, TS, TS)**
- **IF(TS, S, TS)**
- **IF(TS, TS, S)**
- **IF(TS, S, S)**
- **IF(S, TS, TS)**

The following sections provide more details and examples for these syntaxes.

IF(TS *Comparison Operator* S, scalar2 | metric2, scalar3 | metric3)

The corresponding output time series value:

- has the value of **scalar2** or **metric2**, if TS *Comparison Operator* S is TRUE
- has the value of **scalar3** or **metric3**, if TS *Comparison Operator* S is FALSE
- is an empty time series, if the corresponding data point of does not exist in **metric3**, or if **scalar3/metric3** is omitted from the expression

IF(metric1, metric2, metric3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist in **metric3** or if **metric3** is omitted from the expression.

The following table shows an example for this syntax.

| Metric or function | Values |
|-------------------------------|-------------------|
| (metric1) | [1, 1, 0, 0, -] |
| (metric2) | [30, -, 0, 0, 30] |
| (metric3) | [0, 0, 20, -, 20] |
| IF(metric1, metric2, metric3) | [30, 0, 20, -, -] |

IF(metric1, scalar2, metric3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **scalar2**, if the corresponding data point of **metric1** is TRUE.

- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist on **metric3**, or if **metric3** is omitted from the expression.

| Metric or function | Values |
|--------------------------------------|-------------------|
| (metric1) | [1, 1, 0, 0, -] |
| scalar2 | 5 |
| (metric3) | [0, 0, 20, -, 20] |
| IF(metric1, scalar2, metric3) | [5, 5, 20, -, -] |

IF(metric1, metric2, scalar3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **scalar3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped if the corresponding data point in **metric1** does not exist.

| Metric or function | Values |
|--------------------------------------|-------------------|
| (metric1) | [1, 1, 0, 0, -] |
| (metric2) | [30, -, 0, 0, 30] |
| scalar3 | 5 |
| IF(metric1, metric2, scalar3) | [30, 0, 5, 5, -] |

IF(scalar1, metric2, metric3)

The corresponding output time series value:

- has the value of **metric2**, if **scalar1** is TRUE.
- has the value of **metric3**, if **scalar1** is FALSE.
- is an empty time series, if **metric3** is omitted from the expression.

Use case examples for IF expressions

The following examples illustrate the possible uses of the **IF** function.

- To display only the low values of a metric:

IF(metric1<400, metric1)

- To change each data point in a metric to one of two values, to show relative highs and lows of the original metric:

IF(metric1<400, 10, 2)

- To display a 1 for each timestamp where latency is over the threshold, and display a 0 for all other data points:

IF(latency>threshold, 1, 0)

Using metric math with the GetMetricData API operation

You can use [GetMetricData](#) to perform calculations using math expressions, and also retrieve large batches of metric data in one API call. For more information, see [GetMetricData](#).

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions.

Anomaly detection on metric math doesn't support the following functions:

- Expressions that contain more than one **ANOMALY_DETECTION_BAND** in the same line.
- Expressions that contain more than 10 metrics or math expressions.
- Expressions that contain the **METRICS** expression.
- Expressions that contain the **SEARCH** function.
- Expressions that use metrics with different periods.
- Metric math anomaly detectors that use high-resolution metrics as input.

For more information about this feature, see [Using CloudWatch anomaly detection in the Amazon CloudWatch User Guide](#).

Using search expressions in graphs

Search expressions are a type of math expression that you can add to CloudWatch graphs. Search expressions enable you to quickly add multiple related metrics to a graph. They also enable you to create dynamic graphs that automatically add appropriate metrics to their display, even if those metrics don't exist when you first create the graph.

For example, you can create a search expression that displays the `AWS/EC2 CPUUtilization` metric for all instances in the Region. If you later launch a new instance, the `CPUUtilization` of the new instance is automatically added to the graph.

When you use a search expression in a graph, the search finds the search expression in metric names, namespaces, dimension names, and dimension values. You can use Boolean operators for more complex and powerful searches.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

Topics

- [CloudWatch search expression syntax \(p. 120\)](#)

- [CloudWatch search expression examples \(p. 124\)](#)
- [Creating a CloudWatch graph with a search expression \(p. 125\)](#)

CloudWatch search expression syntax

A valid search expression has the following format.

```
SEARCH(' {Namespace, DimensionName1, DimensionName2, ...} SearchTerm', 'Statistic', Period)
```

For example:

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average', 300)
```

- The first part of the query after the word `SEARCH`, enclosed in curly braces, is the *metric schema* to be searched. The metric schema contains a metric namespace and one or more dimension names. Including a metric schema in a search query is optional. If specified, the metric schema must contain a namespace and can optionally contain one or more dimension names that are valid in that namespace.

You don't need to use quote marks inside the metric schema unless a namespace or dimension name includes spaces or non-alphanumeric characters. In that case, you must enclose the name that contains those characters with double quotes.

- The `SearchTerm` is also optional, but a valid search must contain either the metric schema, the `SearchTerm`, or both. The `SearchTerm` usually contains one or more metric names or dimension values. The `SearchTerm` can include multiple terms to search for, by both partial match and exact match. It can also contain Boolean operators.

The `SearchTerm` can include one or more designators, such as `MetricName=` as in this example, but using designators isn't required.

The metric schema and `SearchTerm` must be enclosed together in a pair of single quote marks.

- The `Statistic` is the name of any valid CloudWatch statistic. It must be enclosed by single quotes. For more information, see [Statistics \(p. 5\)](#).
- The `Period` is the aggregation time period in seconds.

The preceding example searches the `AWS/EC2` namespace for any metrics that have `InstanceId` as a dimension name. It returns all `CPUUtilization` metrics that it finds, with the graph showing the `Average` statistic with an aggregation period of 5 minutes.

Search expression limits

The maximum search expression query size is 1024 characters. You can have as many as 100 search expressions on one graph. A graph can display as many as 500 time series.

CloudWatch search expressions: Tokenization

When you specify a `SearchTerm`, the search function searches for *tokens*, which are substrings that CloudWatch automatically generates from full metric names, dimension names, dimension values, and namespaces. CloudWatch generates tokens distinguished by the camel-case capitalization in the original string. Numeric characters also serve as the start of new tokens, and non-alphanumeric characters serve as delimiters, creating tokens before and after the non-alphanumeric characters.

A continuous string of the same type of token delimiter character results in one token.

All generated tokens are in lowercase. The following table shows some examples of tokens generated.

| Original string | Tokens generated |
|-------------------|--|
| CustomCount1 | customcount1, custom, count, 1 |
| SDBFailure | sdbfailure, sdb, failure |
| Project2-trial333 | project2trial333, project, 2, trial, 333 |

CloudWatch search expressions: Partial matches

When you specify a `SearchTerm`, the search term is also tokenized. CloudWatch finds metrics based on partial matches, which are matches of a single token generated from the search term to a single token generated from a metric name, namespace, dimension name, or dimension value.

Partial match searches to match a single token are case insensitive. For example, using any of the following search terms can return the `CustomCount1` metric:

- `count`
- `Count`
- `COUNT`

However, using `couNT` as a search term doesn't find `CustomCount1` because the capitalization in the search term `couNT` is tokenized into `cou` and `NT`.

Searches can also match composite tokens, which are multiple tokens that appear consecutively in the original name. To match a composite token, the search is case sensitive. For example, if the original term is `CustomCount1`, searches for `CustomCount` or `Count1` are successful, but searches for `customcount` or `count1` aren't.

CloudWatch search expressions: Exact matches

You can define a search to find only exact matches of your search term by using double quotes around the part of the search term that requires an exact match. These double-quotes are enclosed in the single-quotes used around the entire search term. For example, `SEARCH(' {MyNamespace}, "CustomCount1" ', 'Maximum', 120)` finds the exact string `CustomCount1` if it exists as a metric name, dimension name, or dimension value in the namespace named `MyNamespace`. However, the searches `SEARCH(' {MyNamespace}, "customcount1" ', 'Maximum', 120)` or `SEARCH(' {MyNamespace}, "Custom" ', 'Maximum', 120)` do not find this string.

You can combine partial match terms and exact match terms in a single search expression. For example, `SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum', 120)` returns the Elastic Load Balancing metric named `ConsumedLCUs` as well as all Elastic Load Balancing metrics or dimensions that contain the token `flow`.

Using exact match is also a good way to find names with special characters, such as non-alphanumeric characters or spaces, as in the following example.

```
SEARCH(' {"My Namespace", "Dimension@Name"}, "Custom:Name[Special_Characters" ', 'Maximum',
120)
```

CloudWatch search expressions: Excluding a metric schema

All examples shown so far include a metric schema, in curly braces. Searches that omit a metric schema are also valid.

For example, `SEARCH(' "CPUutilization" ', 'Average', 300)` returns all metric names, dimension names, dimension values, and namespaces that are an exact match for the string `CPUUtilization`. In the AWS metric namespaces, this can include metrics from several services including Amazon EC2, Amazon ECS, SageMaker, and others.

To narrow this search to only one AWS service, the best practice is to specify the namespace and any necessary dimensions in the metric schema, as in the following example. Although this narrows the search to the AWS/EC2 namespace, it would still return results of other metrics if you have defined `CPUUtilization` as a dimension value for those metrics.

```
SEARCH(' {AWS/EC2, InstanceType} "CPUUtilization" ', 'Average', 300)
```

Alternatively you could add the namespace in the `SearchTerm` as in the following example. But in this example, the search would match any AWS/EC2 string, even if it was a custom dimension name or value.

```
SEARCH(' "AWS/EC2" MetricName="CPUUtilization" ', 'Average', 300)
```

CloudWatch search expressions: Specifying property names in the search

The following exact match search for `"CustomCount1"` returns all metrics with exactly that name.

```
SEARCH(' "CustomCount1" ', 'Maximum', 120)
```

But it also returns metrics with dimension names, dimension values, or namespaces of `CustomCount1`. To structure your search further, you can specify the property name of the type of object that you want to find in your searches. The following example searches all namespaces and returns metrics named `CustomCount1`.

```
SEARCH(' MetricName="CustomCount1" ', 'Maximum', 120)
```

You can also use namespaces and dimension name/value pairs as property names, as in the following examples. The first of these examples also illustrates that you can use property names with partial match searches as well.

```
SEARCH(' InstanceType=micro ', 'Average', 300)
```

```
SEARCH(' InstanceType="t2.micro" Namespace="AWS/EC2" ', 'Average', 300)
```

CloudWatch search expressions: Non-alphanumeric characters

Non-alphanumeric characters serve as delimiters, and mark where the names of metrics, dimensions, namespaces, and search terms are to be separated into tokens. When terms are tokenized, non-alphanumeric characters are stripped out and don't appear in the tokens. For example, `Network-Errors_2` generates the tokens `network`, `errors`, and `2`.

Your search term can include any non-alphanumeric characters. If these characters appear in your search term, they can specify composite tokens in a partial match. For example, all of the following searches would find metrics named either `Network-Errors-2` or `NetworkErrors2`.

```
network/errors
network+errors
network-errors
```

Network_Errors

When you're doing an exact value search, any non-alphanumeric characters used in the exact search must be the correct characters that appear in the string being searched for. For example, if you want to find Network-Errors-2, searching for "Network-Errors-2" is successful, but a search for "Network_Errors_2" isn't.

When you perform an exact match search, the following characters must be escaped with a backslash.

" \ ()

For example, to find the metric name Europe\France Traffic(Network) by exact match, use the search term "Europe\\France Traffic\\(Network\\)"

CloudWatch search expressions: Boolean operators

Search supports the use of the Boolean operators AND, OR, and NOT within the SearchTerm. Boolean operators are enclosed in the single quote marks that you use to enclose the entire search term. Boolean operators are case sensitive, so and, or, and not aren't valid as Boolean operators.

You can use AND explicitly in your search, such as `SEARCH('{AWS/EC2,InstanceId} network AND packets', 'Average', 300)`. Not using any Boolean operator between search terms implicitly searches them as if there were an AND operator, so `SEARCH(' {AWS/EC2,InstanceId} network packets ', 'Average', 300)` yields the same search results.

Use NOT to exclude subsets of data from the results. For example, `SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT i-1234567890123456 ', 'Average', 300)` returns the CPUUtilization for all your instances, except for the instance i-1234567890123456. You can also use a NOT clause as the only search term. For example, `SEARCH('NOT Namespace=AWS ', 'Maximum', 120)` yields all your custom metrics (metrics with namespaces that don't include AWS).

You can use multiple NOT phrases in a query. For example, `SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT "ProjectA" NOT "ProjectB" ', 'Average', 300)` returns the CPUUtilization of all instances in the Region, except for those with dimension values of ProjectA or ProjectB.

You can combine Boolean operators for more powerful and detailed searches, as in the following examples. Use parentheses to group the operators.

Both of the next two examples return all metric names containing ReadOps from both the EC2 and EBS namespaces.

`SEARCH(' (EC2 OR EBS) AND MetricName=ReadOps ', 'Maximum', 120)`

`SEARCH(' (EC2 OR EBS) MetricName=ReadOps ', 'Maximum', 120)`

The following example narrows the previous search to only results that include ProjectA, which could be the value of a dimension.

`SEARCH(' (EC2 OR EBS) AND ReadOps AND ProjectA ', 'Maximum', 120)`

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions, and Invocations of functions with names that include the strings ProjectA or ProjectB.

`SEARCH(' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND (ProjectA OR ProjectB)) ', 'Average', 600)`

CloudWatch search expressions: Using math expressions

You can use a search expression within a math expressions in a graph.

For example, `SUM(SEARCH(' {AWS/Lambda, FunctionName} MetricName="Errors" ', 'Sum', 300))` returns the sum of the Errors metric of all your Lambda functions.

Using separate lines for your search expression and math expression might yield more useful results. For example, suppose that you use the following two expressions in a graph. The first line displays separate Errors lines for each of your Lambda functions. The ID of this expression is e1. The second line adds another line showing the sum of the errors from all of the functions.

```
SEARCH(' {AWS/Lambda, FunctionName}, MetricName="Errors" ', 'Sum', 300)  
SUM(e1)
```

CloudWatch search expression examples

The following examples illustrate more search expression uses and syntax. Let's start with a search for `CPUUtilization` across all instances in the Region and then look at variations.

This example displays one line for each instance in the Region, showing the `CPUUtilization` metric from the `AWS/EC2` namespace.

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average', 300)
```

Changing `InstanceId` to `InstanceType` changes the graph to show one line for each instance type used in the Region. Data from all instances of each type is aggregated into one line for that instance type.

```
SEARCH(' {AWS/EC2,InstanceType} MetricName="CPUUtilization" ', 'Average', 300)
```

Removing the dimension name but keeping the namespace in the schema, as in the following example, results in a single line showing the aggregation of `CPUUtilization` metrics for all instances in the Region.

```
SEARCH(' {AWS/EC2} MetricName="CPUUtilization" ', 'Average', 300)
```

The following example aggregates the `CPUUtilization` by instance type and displays one line for each instance type that includes the string `micro`.

```
SEARCH(' {AWS/EC2,InstanceType} InstanceType=micro MetricName="CPUUtilization" ',  
'Average', 300)
```

This example narrows the previous example, changing the `InstanceType` to an exact search for `t2.micro` instances.

```
SEARCH(' {AWS/EC2,InstanceType} InstanceType="t2.micro" MetricName="CPUUtilization" ',  
'Average', 300)
```

The following search removes the `{metric schema}` part of the query, so the `CPUUtilization` metric from all namespaces appears in the graph. This can return quite a few results because the graph includes multiple lines for the `CPUUtilization` metric from each AWS service, aggregated along different dimensions.

```
SEARCH(' MetricName="CPUUtilization" ', 'Average', 300)
```

To narrow these results a bit, you can specify two specific metric namespaces.

```
SEARCH(' MetricName="CPUUtilization" AND ("AWS/ECS" OR "AWS/ES") ', 'Average', 300)
```

The preceding example is the only way to do a search of specific multiple namespaces with one search query, as you can specify only one metric schema in each query. However, to add more structure, you could use two queries in the graph, as in the following example. This example also adds more structure by specifying a dimension to use to aggregate the data for Amazon ECS.

```
SEARCH(' {AWS/ECS, ClusterName}, MetricName="CPUUtilization" ', 'Average', 300)  
SEARCH(' {AWS/EBS}, MetricName="CPUUtilization" ', 'Average', 300)
```

The following example returns the Elastic Load Balancing metric named ConsumedLCUs as well as all Elastic Load Balancing metrics or dimensions that contain the token flow.

```
SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum', 120)
```

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions and Invocations of functions with names that include the strings ProjectA or ProjectB.

```
SEARCH(' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND (ProjectA OR ProjectB)) ', 'Average', 600)
```

The following example displays all of your custom metrics, excluding metrics generated by AWS services.

```
SEARCH(' NOT Namespace=AWS ', 'Average', 120)
```

The following example displays metrics with metric names, namespaces, dimension names, and dimension values that contain the string Errors as part of their name.

```
SEARCH(' Errors ', 'Average', 300)
```

The following example narrows that search to exact matches. For example, this search finds the metric name Errors but not metrics named ConnectionErrors or errors.

```
SEARCH(' "Errors" ', 'Average', 300)
```

The following example shows how to specify names that contain spaces or special characters in the metric schema part of the search term.

```
SEARCH(' {"Custom-Namespace", "Dimension Name With Spaces"}, ErrorCount ', 'Maximum', 120)
```

Creating a CloudWatch graph with a search expression

On the CloudWatch console, you can access search capability when you add a graph to a dashboard, or by using the **Metrics** view.

You can't create an alarm based on a **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To add a graph with a search expression to an existing dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose either **Line** or **Stacked area** and choose **Configure**.
5. On the **Graphed metrics** tab, choose **Add a math expression**.
6. For **Details**, enter the search expression that you want. For example, `SEARCH('AWS/EC2,InstanceId} MetricName="CPUUtilization", 'Average', 300)`
7. (Optional) To add another search expression or math expression to the graph, choose **Add a math expression**
8. (Optional) After you add a search expression, you can specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics** and then **Dynamic labels**.

By default, the dynamic values you add to the label appear at the beginning of the label. You can then click the **Label** value for the metric to edit the label. For more information, see [Using dynamic labels \(p. 88\)](#).

9. (Optional) To add a single metric to the graph, choose the **All metrics** tab and drill down to the metric you want.
10. (Optional) To change the time range shown on the graph, choose either **custom** at the top of the graph or one of the time periods to the left of **custom**.
11. (Optional) Horizontal annotations help dashboard users quickly see when a metric has spiked to a certain level or whether the metric is within a predefined range. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.
 - b. For **Value**, enter the metric value where the horizontal annotation appears.
 - c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose **Above** or **Below** for the corresponding area to be filled. If you specify **Between**, another **Value** field appears, and the area of the graph between the two values is filled.
 - d. For **Axis**, specify whether the numbers in **value** refer to the metric associated with the left y-axis or the right y-axis if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

12. (Optional) Vertical annotations help you mark milestones in a graph, such as operational events or the beginning and end of a deployment. To add a vertical annotation, choose **Graph options** and then **Add vertical annotation**:
 - a. For **Label**, enter a label for the annotation. To show only the date and time on the annotation, keep the **Label** field blank.
 - b. For **Date**, specify the date and time where the vertical annotation appears.

- c. For **Fill**, specify whether to use fill shading before or after a vertical annotation or between two vertical annotations. For example, choose **Before** or **After** for the corresponding area to be filled. If you specify **Between**, another Date field appears, and the area of the graph between the two values is filled.

Repeat these steps to add multiple vertical annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

13. Choose **Create widget**.
14. Choose **Save dashboard**.

To use the Metrics view to graph searched metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the search field, enter the tokens to search for: for example, `cpututilization t2.small`.

Results that match your search appear.

4. To graph all of the metrics that match your search, choose **Graph search**.

or

To refine your search, choose one of the namespaces that appeared in your search results.

5. If you selected a namespace to narrow your results, you can do the following:
 - a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
 - b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
 - c. To view help for a metric, select the metric name and choose **What is this?**.

The selected metrics appear on the graph.

6. (Optional) Select one of the buttons in the search bar to edit that part of the search term.
7. (Optional) To add the graph to a dashboard, choose **Actions** and then **Add to dashboard**.

Use metrics explorer to monitor resources by their tags and properties

Metrics explorer is a tag-based tool that enables you to filter, aggregate, and visualize your metrics by tags and resource properties, to enhance observability for your services. This gives you a flexible and dynamic troubleshooting experience, so that you can create multiple graphs at a time and use these graphs to build your application health dashboards.

Metrics explorer visualizations are dynamic, so if a matching resource is created after you create a metrics explorer widget and add it to a CloudWatch dashboard, the new resource automatically appears in the explorer widget.

For example, if all of your EC2 production instances have the **production** tag, you can use metrics explorer to filter and aggregate metrics from all of these instances to understand their health and performance. If a new instance with a matching tag is later created, it's automatically added to the metrics explorer widget.

With metrics explorer, you can choose how to aggregate metrics from the resources that match the criteria, and whether to show them all in a single graph or on different graphs within one metrics explorer widget.

Metrics explorer includes templates that you can use to see useful visualization graphs with one click, and you can also extend these templates to create completely customized metrics explorer widgets.

Metrics explorer supports metrics emitted by AWS and EC2 metrics that are published by the CloudWatch agent, including memory, disk, and CPU metrics. To use metrics explorer to see the metrics that are published by the CloudWatch agent, you might have to update your CloudWatch agent configuration file. For more information, see [CloudWatch agent configuration for metrics explorer \(p. 129\)](#)

To create a visualization with metrics explorer and optionally add it to a dashboard, follow these steps.

To create a visualization with metrics explorer

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Explorer**.
3. Do one of the following:
 - To use a template, select it in the box that currently shows **Empty Explorer**.

Depending on the template, the explorer might immediately display graphs of metrics. If it doesn't, choose one or more tags or properties in the **From** box and then data should appear. If it doesn't, use the options at the top of the page to display a longer time range in the graphs.

- To create a custom visualization, under **Metrics**, choose a single metric or all the available metrics from a service.

After you choose a metric, you can optionally repeat this step to add more metrics.

4. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name, and then choose the statistic that you want.
5. Under **From**, choose a tag or a resource property to filter your results.

After you do this, you can optionally repeat this step to choose more tags or resource properties.

If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

If you choose different properties or tags, such as the **Production** tag and the M5 instance type, only the resources that match all of these selections are displayed. It's treated as an AND operation.

6. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

Depending on how you choose to aggregate, the result may be a single time series or multiple time series.

7. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
8. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.

9. To add this visualization as a widget to a CloudWatch dashboard, choose **Add to dashboard**.

CloudWatch agent configuration for metrics explorer

To enable metrics explorer to discover EC2 metrics published by the CloudWatch agent, make sure that the CloudWatch agent configuration file contains the following values:

- In the `metrics` section, make sure that the `aggregation_dimensions` parameter includes `["InstanceId"]`. It can also contain other dimensions.
- In the `metrics` section, make sure that the `append_dimensions` parameter includes a `{ "InstanceId": "${aws:InstanceId}" }` line. It can also contain other lines.
- In the `metrics` section, inside the `metrics_collected` section, check the sections for each resource type that you want metrics explorer to discover, such as the `cpu`, `disk`, and `memory` sections. Make sure that each of these sections has a `"resources": ["*"]` line..
- In the `cpu` section of the `metrics_collected` section, make sure there is a `"totalcpu": true` line.
- You must use the default `cwagent` namespace for the metrics collected by the CloudWatch agent, instead of a custom namespace.

The settings in the previous list cause the CloudWatch agent to publish aggregate metrics for disks, CPUs, and other resources that can be plotted in metrics explorer for all the instances that use it.

These settings will republish the metrics that you had previously set up to be published with multiple dimensions, adding to your metric costs.

For more information about editing the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

Using Amazon CloudWatch alarms

You can create both *metric alarms* and *composite alarms* in CloudWatch.

- A *metric alarm* watches a single CloudWatch metric or the result of a math expression based on CloudWatch metrics. The alarm performs one or more actions based on the value of the metric or expression relative to a threshold over a number of time periods. The action can be sending a notification to an Amazon SNS topic, performing an Amazon EC2 action or an Amazon EC2 Auto Scaling action, or creating an OpsItem or incident in Systems Manager.
- A *composite alarm* includes a rule expression that takes into account the alarm states of other alarms that you have created. The composite alarm goes into ALARM state only if all conditions of the rule are met. The alarms specified in a composite alarm's rule expression can include metric alarms and other composite alarms.

Using composite alarms can reduce alarm noise. You can create multiple metric alarms, and also create a composite alarm and set up alerts only for the composite alarm. For example, a composite might go into ALARM state only when all of the underlying metric alarms are in ALARM state.

Composite alarms can send Amazon SNS notifications when they change state, and can create Systems Manager OpsItems or incidents when they go into ALARM state, but can't perform EC2 actions or Auto Scaling actions.

There is no limit on the number of alarms that you create in your account.

You can add alarms to CloudWatch dashboards and monitor them visually. When an alarm is on a dashboard, it turns red when it is in the ALARM state, making it easier for you to monitor its status proactively.

An alarm invokes actions only when the alarm changes state. The exception is for alarms with Auto Scaling actions. For Auto Scaling actions, the alarm continues to invoke the action once per minute that the alarm remains in the new state.

An alarm can watch a metric in the same account. If you have enabled cross-account functionality in your CloudWatch console, you can also create alarms that watch metrics in other AWS accounts. Creating cross-account composite alarms is not supported. Creating cross-account alarms that use math expressions is supported, except that the ANOMALY_DETECTION_BAND, INSIGHT_RULE, and SERVICE_QUOTA functions are not supported for cross-account alarms.

Note

CloudWatch doesn't test or validate the actions that you specify, nor does it detect any Amazon EC2 Auto Scaling or Amazon SNS errors resulting from an attempt to invoke nonexistent actions. Make sure that your alarm actions exist.

Metric alarm states

A metric alarm has the following possible states:

- **OK** – The metric or expression is within the defined threshold.
- **ALARM** – The metric or expression is outside of the defined threshold.

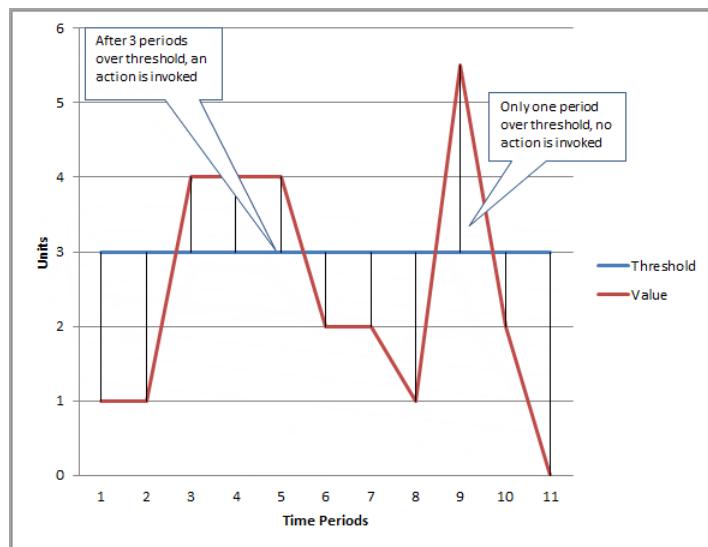
- **INSUFFICIENT_DATA** – The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

Evaluating an alarm

When you create an alarm, you specify three settings to enable CloudWatch to evaluate when to change the alarm state:

- **Period** is the length of time to evaluate the metric or expression to create each individual data point for an alarm. It is expressed in seconds. If you choose one minute as the period, the alarm evaluates the metric once per minute.
- **Evaluation Periods** is the number of the most recent periods, or data points, to evaluate when determining alarm state.
- **Datapoints to Alarm** is the number of data points within the Evaluation Periods that must be breaching to cause the alarm to go to the **ALARM** state. The breaching data points don't have to be consecutive, but they must all be within the last number of data points equal to **Evaluation Period**.

In the following figure, the alarm threshold for a metric alarm is set to three units. Both **Evaluation Period** and **Datapoints to Alarm** are 3. That is, when all existing data points in the most recent three consecutive periods are above the threshold, the alarm goes to **ALARM** state. In the figure, this happens in the third through fifth time periods. At period six, the value dips below the threshold, so one of the periods being evaluated is not breaching, and the alarm state changes back to **OK**. During the ninth time period, the threshold is breached again, but for only one period. Consequently, the alarm state remains **OK**.



When you configure **Evaluation Periods** and **Datapoints to Alarm** as different values, you're setting an "M out of N" alarm. **Datapoints to Alarm** is ("M") and **Evaluation Periods** is ("N"). The evaluation interval is the number of data points multiplied by the period. For example, if you configure 4 out of 5 data points with a period of 1 minute, the evaluation interval is 5 minutes. If you configure 3 out of 3 data points with a period of 10 minutes, the evaluation interval is 30 minutes.

Note

If data points are missing soon after you create an alarm, and the metric was being reported to CloudWatch before you created the alarm, CloudWatch retrieves the most recent data points from before the alarm was created when evaluating the alarm.

Alarm actions

You can specify what actions an alarm takes when it changes state between the OK, ALARM, and INSUFFICIENT_DATA states. The most common type of alarm action is to notify one or more people by sending a message to an Amazon Simple Notification Service topic. For more information about Amazon SNS, see [What is Amazon SNS?](#).

Alarms based on EC2 metrics can also perform EC2 actions, such as stopping, terminating, rebooting, or recovering an EC2 instance. For more information, see [Create alarms to stop, terminate, reboot, or recover an EC2 instance \(p. 154\)](#).

Alarms can also perform actions to scale an Auto Scaling group. For more information, see [Step and simple scaling policies for Amazon EC2 Auto Scaling](#).

You can also configure alarms to create OpsItems in Systems Manager Ops Center or create incidents in AWS Systems Manager Incident Manager. These actions are performed only when the alarm goes into ALARM state. For more information, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Configuring how CloudWatch alarms treat missing data

Sometimes, not every expected data point for a metric gets reported to CloudWatch. For example, this can happen when a connection is lost, a server goes down, or when a metric reports data only intermittently by design.

CloudWatch enables you to specify how to treat missing data points when evaluating an alarm. This helps you to configure your alarm so that it goes to ALARM state only when appropriate for the type of data being monitored. You can avoid false positives when missing data doesn't indicate a problem.

Similar to how each alarm is always in one of three states, each specific data point reported to CloudWatch falls under one of three categories:

- Not breaching (within the threshold)
- Breaching (violating the threshold)
- Missing

For each alarm, you can specify CloudWatch to treat missing data points as any of the following:

- `notBreaching` – Missing data points are treated as "good" and within the threshold
- `breaching` – Missing data points are treated as "bad" and breaching the threshold
- `ignore` – The current alarm state is maintained
- `missing` – If all data points in the alarm evaluation range are missing, the alarm transitions to INSUFFICIENT_DATA.

The best choice depends on the type of metric. For a metric that continually reports data, such as `CPUUtilization` of an instance, you might want to treat missing data points as `breaching`, because they might indicate that something is wrong. But for a metric that generates data points only when an error occurs, such as `ThrottledRequests` in Amazon DynamoDB, you would want to treat missing data as `notBreaching`. The default behavior is `missing`.

Choosing the best option for your alarm prevents unnecessary and misleading alarm condition changes, and also more accurately indicates the health of your system.

Important

Alarms that evaluate metrics in the AWS/DynamoDB namespace always ignore missing data even if you choose a different option for how the alarm should treat missing data. When an AWS/DynamoDB metric has missing data, alarms that evaluate that metric remain in their current state.

How alarm state is evaluated when data is missing

Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as **Evaluation Periods**. The exact number of data points it attempts to retrieve depends on the length of the alarm period and whether it is based on a metric with standard resolution or high resolution. The time frame of the data points that it attempts to retrieve is the *evaluation range*.

Once CloudWatch retrieves these data points, the following happens:

- If no data points in the evaluation range are missing, CloudWatch evaluates the alarm based on the most recent data points collected. The number of data points evaluated is equal to the **Evaluation Periods** for the alarm. The extra data points from farther back in the evaluation range are not needed and are ignored.
- If some data points in the evaluation range are missing, but the total number of existing data points that were successfully retrieved from the evaluation range is equal to or more than the alarm's **Evaluation Periods**, CloudWatch evaluates the alarm state based on the most recent real data points that were successfully retrieved, including the necessary extra data points from farther back in the evaluation range. In this case, the value you set for how to treat missing data is not needed and is ignored.
- If some data points in the evaluation range are missing, and the number of actual data points that were retrieved is lower than the alarm's number of **Evaluation Periods**, CloudWatch fills in the missing data points with the result you specified for how to treat missing data, and then evaluates the alarm. However, all real data points in the evaluation range are included in the evaluation. CloudWatch uses missing data points only as few times as possible.

Note

A particular case of this behavior is that CloudWatch alarms might repeatedly re-evaluate the last set of data points for a period of time after the metric has stopped flowing. This re-evaluation might cause the alarm to change state and re-execute actions, if it had changed state immediately prior to the metric stream stopping. To mitigate this behavior, use shorter periods.

The following tables illustrate examples of the alarm evaluation behavior. In the first table, **Datapoints to Alarm** and **Evaluation Periods** are both 3. CloudWatch retrieves the 5 most recent data points when evaluating the alarm, in case some of the most recent 3 data points are missing. 5 is the evaluation range for the alarm.

Column 1 shows the 5 most recent data points, because the evaluation range is 5. These data points are shown with the most recent data point on the right. 0 is a non-breaching data point, X is a breaching data point, and - is a missing data point.

Column 2 shows how many of the 3 necessary data points are missing. Even though the most recent 5 data points are evaluated, only 3 (the setting for **Evaluation Periods**) are necessary to evaluate the alarm state. The number of data points in Column 2 is the number of data points that must be "filled in", using the setting for how missing data is being treated.

In columns 3-6, the column headers are the possible values for how to treat missing data. The rows in these columns show the alarm state that is set for each of these possible ways to treat missing data.

| Data points | # of data points that must be filled | MISSING | IGNORE | BREACHING | NOT BREACHING |
|-------------|--------------------------------------|-------------------|----------------------|-----------|---------------|
| 0 - X - X | 0 | OK | OK | OK | OK |
| 0 - - - | 2 | OK | OK | OK | OK |
| - - - - | 3 | INSUFFICIENT DATA | Retain current state | ALARM | OK |
| 0 X X - X | 0 | ALARM | ALARM | ALARM | ALARM |
| -- X - - | 2 | ALARM | Retain current state | ALARM | OK |

In the second row of the preceding table, the alarm stays **OK** even if missing data is treated as breaching, because the one existing data point is not breaching, and this is evaluated along with two missing data points which are treated as breaching. The next time this alarm is evaluated, if the data is still missing it will go to **ALARM**, as that non-breaching data point will no longer be in the evaluation range.

The third row, where all five of the most recent data points are missing, illustrates how the various settings for how to treat missing data affect the alarm state. If missing data points are considered breaching, the alarm goes into **ALARM** state, while if they are considered not breaching, then the alarm goes into **OK** state. If missing data points are ignored, the alarm retains the current state it had before the missing data points. And if missing data points are just considered as missing, then the alarm does not have enough recent real data to make an evaluation, and goes into **INSUFFICIENT_DATA**.

In the fourth row, the alarm goes to **ALARM** state in all cases because the three most recent data points are breaching, and the alarm's **Evaluation Periods** and **Datapoints to Alarm** are both set to 3. In this case, the missing data point is ignored and the setting for how to evaluate missing data is not needed, because there are 3 real data points to evaluate.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see [Avoiding premature transitions to alarm state \(p. 135\)](#).

In the next table, the **Period** is again set to 5 minutes, and **Datapoints to Alarm** is only 2 while **Evaluation Periods** is 3. This is a 2 out of 3, M out of N alarm.

The evaluation range is 5. This is the maximum number of recent data points that are retrieved and can be used in case some data points are missing.

| Data points | # of missing data points | MISSING | IGNORE | BREACHING | NOT BREACHING |
|-------------|--------------------------|---------|----------------------|-----------|---------------|
| 0 - X - X | 0 | ALARM | ALARM | ALARM | ALARM |
| 0 0 X 0 X | 0 | ALARM | ALARM | ALARM | ALARM |
| 0 - X - - | 1 | OK | OK | ALARM | OK |
| - - - 0 | 2 | OK | OK | ALARM | OK |
| - - - X - | 2 | ALARM | Retain current state | ALARM | OK |

In rows 1 and 2, the alarm always goes to ALARM state because 2 of the 3 most recent data points are breaching. In row 2, the two oldest data points in the evaluation range are not needed because none of the 3 most recent data points are missing, so these two older data points are ignored.

In rows 3 and 4, the alarm goes to ALARM state only if missing data is treated as breaching, in which case the two most recent missing data points are both treated as breaching. In row 4, these two missing data points that are treated as breaching provide the two necessary breaching data points to trigger the ALARM state.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see the following section.

Avoiding premature transitions to alarm state

CloudWatch alarm evaluation includes logic to try to avoid false alarms, where the alarm goes into ALARM state prematurely when data is intermittent. The example shown in row 5 in the tables in the previous section illustrate this logic. In those rows, and in the following examples, the **Evaluation Periods** is 3 and the evaluation range is 5 data points. **Datapoints to Alarm** is 3, except for the M out of N example, where **Datapoints to Alarm** is 2.

Suppose an alarm's most recent data is - - - - x, with four missing data points and then a breaching data point as the most recent data point. Because the next data point may be non-breaching, the alarm does not go immediately into ALARM state when the data is either - - - - x or - - - x - and **Datapoints to Alarm** is 3. This way, false positives are avoided when the next data point is non-breaching and causes the data to be - - x 0 or - - x - 0.

However, if the last few data points are - - x - -, the alarm goes into ALARM state even if missing data points are treated as missing. This is because alarms are designed to always go into ALARM state when the oldest available breaching datapoint during the **Evaluation Periods** number of data points is at least as old as the value of **Datapoints to Alarm**, and all other more recent data points are breaching or missing. In this case, the alarm goes into ALARM state even if the total number of datapoints available is lower than M (**Datapoints to Alarm**).

This alarm logic applies to M out of N alarms as well. If the oldest breaching data point during the evaluation range is at least as old as the value of **Datapoints to Alarm**, and all of the more recent data points are either breaching or missing, the alarm goes into ALARM state no matter the value of M (**Datapoints to Alarm**).

High-resolution alarms

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see [Publishing custom metrics \(p. 94\)](#).

Alarms on math expressions

You can set an alarm on the result of a math expression that is based on one or more CloudWatch metrics. A math expression used for an alarm can include as many as 10 metrics. Each metric must be using the same period.

For an alarm based on a math expression, you can specify how you want CloudWatch to treat missing data points for the underlying metrics when evaluating the alarm.

Alarms based on math expressions can't perform Amazon EC2 actions.

For more information about metric math expressions and syntax, see [Using metric math \(p. 97\)](#).

Percentile-based CloudWatch alarms and low data samples

When you set a percentile as the statistic for an alarm, you can specify what to do when there is not enough data for a good statistical assessment. You can choose to have the alarm evaluate the statistic anyway and possibly change the alarm state. Or, you can have the alarm ignore the metric while the sample size is low, and wait to evaluate it until there is enough data to be statistically significant.

For percentiles between 0.5 (inclusive) and 1.00 (exclusive), this setting is used when there are fewer than $10/(1\text{-percentile})$ data points during the evaluation period. For example, this setting would be used if there were fewer than 1000 samples for an alarm on a p99 percentile. For percentiles between 0 and 0.5 (exclusive), the setting is used when there are fewer than $10/\text{percentile}$ data points.

CloudWatch alarms and Amazon EventBridge

CloudWatch sends events to Amazon EventBridge whenever a CloudWatch alarm changes alarm state. You can use these alarm state change events to trigger an event target in EventBridge. For more information, see [Alarm events and EventBridge \(p. 753\)](#).

Common features of CloudWatch alarms

The following features apply to all CloudWatch alarms:

- There is no limit to the number of alarms that you can create. To create or update an alarm, you use the CloudWatch console, the [PutMetricAlarm](#) API action, or the [put-metric-alarm](#) command in the AWS CLI.
- Alarm names must contain only ASCII characters.
- You can list any or all of the currently configured alarms, and list any alarms in a particular state by using the CloudWatch console, the [DescribeAlarms](#) API action, or the [describe-alarms](#) command in the AWS CLI.
- You can disable and enable alarms by using the [DisableAlarmActions](#) and [EnableAlarmActions](#) API actions, or the [disable-alarm-actions](#) and [enable-alarm-actions](#) commands in the AWS CLI.
- You can test an alarm by setting it to any state using the [SetAlarmState](#) API action or the [set-alarm-state](#) command in the AWS CLI. This temporary state change lasts only until the next alarm comparison occurs.
- You can create an alarm for a custom metric before you've created that custom metric. For the alarm to be valid, you must include all of the dimensions for the custom metric in addition to the metric namespace and metric name in the alarm definition. To do this, you can use the [PutMetricAlarm](#) API action, or the [put-metric-alarm](#) command in the AWS CLI.
- You can view an alarm's history using the CloudWatch console, the [DescribeAlarmHistory](#) API action, or the [describe-alarm-history](#) command in the AWS CLI. CloudWatch preserves alarm history for two weeks. Each state transition is marked with a unique timestamp. In rare cases, your history might show more than one notification for a state change. The timestamp enables you to confirm unique state changes.
- The number of evaluation periods for an alarm multiplied by the length of each evaluation period can't exceed one day.

Note

Some AWS resources don't send metric data to CloudWatch under certain conditions. For example, Amazon EBS might not send metric data for an available volume that is not attached to an Amazon EC2 instance, because there is no metric activity to be monitored for that volume. If you have an alarm set for such a metric, you might notice its state change to `INSUFFICIENT_DATA`. This might indicate that your resource is inactive, and might not necessarily mean that there is a problem. You can specify how each alarm treats missing data. For more information, see [Configuring how CloudWatch alarms treat missing data \(p. 132\)](#).

Setting up Amazon SNS notifications

Amazon CloudWatch uses Amazon SNS to send email. First, create and subscribe to an SNS topic. When you create a CloudWatch alarm, you can add this SNS topic to send an email notification when the alarm changes state. For more information, see the [Amazon Simple Notification Service Getting Started Guide](#).

Alternatively, if you plan to create your CloudWatch alarm using the AWS Management Console, you can skip this procedure because you can create the topic when you create the alarm.

Note

When you create an Amazon SNS topic, you choose to make it a *standard topic* or a *FIFO topic*. CloudWatch guarantees the publication of all alarm notifications to both types of topics. However, even if you use a FIFO topic, in rare cases CloudWatch sends the notifications to the topic out of order. If you use a FIFO topic, the alarm sets the message group ID of the alarm notifications to be a hash of the ARN of the alarm.

Setting up an Amazon SNS topic using the AWS Management Console

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic.

To create an SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. On the Amazon SNS dashboard, under **Common actions**, choose **Create Topic**.
3. In the **Create new topic** dialog box, for **Topic name**, enter a name for the topic (for example, `my-topic`).
4. Choose **Create topic**.
5. Copy the **Topic ARN** for the next task (for example, `arn:aws:sns:us-east-1:111122223333:my-topic`).

To subscribe to an SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose **Subscriptions**, **Create subscription**.
3. In the **Create subscription** dialog box, for **Topic ARN**, paste the topic ARN that you created in the previous task.
4. For **Protocol**, choose **Email**.
5. For **Endpoint**, enter an email address that you can use to receive the notification, and then choose **Create subscription**.
6. From your email application, open the message from AWS Notifications and confirm your subscription.

Your web browser displays a confirmation response from Amazon SNS.

To publish a test message to an SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic and choose **Publish to topic**.
4. In the **Publish a message** page, for **Subject**, enter a subject line for your message, and for **Message**, enter a brief message.
5. Choose **Publish Message**.
6. Check your email to confirm that you received the message.

Setting up an SNS topic using the AWS CLI

First, you create an SNS topic, and then you publish a message directly to the topic to test that you have properly configured it.

To set up an SNS topic

1. Create the topic using the [create-topic](#) command as follows.

```
aws sns create-topic --name my-topic
```

Amazon SNS returns a topic ARN with the following format:

```
{  
    "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic"  
}
```

2. Subscribe your email address to the topic using the [subscribe](#) command. If the subscription request succeeds, you receive a confirmation email message.

```
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:111122223333:my-topic --protocol  
email --notification-endpoint my-email-address
```

Amazon SNS returns the following:

```
{  
    "SubscriptionArn": "pending confirmation"  
}
```

3. From your email application, open the message from AWS Notifications and confirm your subscription.

Your web browser displays a confirmation response from Amazon Simple Notification Service.

4. Check the subscription using the [list-subscriptions-by-topic](#) command.

```
aws sns list-subscriptions-by-topic --topic-arn arn:aws:sns:us-east-1:111122223333:my-  
topic
```

Amazon SNS returns the following:

```
{  
    "Subscriptions": [  
        {
```

```
        "Owner": "111122223333",
        "Endpoint": "me@mycompany.com",
        "Protocol": "email",
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic",
        "SubscriptionArn": "arn:aws:sns:us-east-1:111122223333:my-topic:64886986-
bf10-48fb-a2f1-dab033aa67a3"
    }
}
```

5. (Optional) Publish a test message to the topic using the [publish](#) command.

```
aws sns publish --message "Verification" --topic arn:aws:sns:us-east-1:111122223333:my-
topic
```

Amazon SNS returns the following.

```
{
    "MessageId": "42f189a0-3094-5cf6-8fd7-c2dde61a4d7d"
}
```

6. Check your email to confirm that you received the message.

Create a CloudWatch alarm based on a static threshold

You choose a CloudWatch metric for the alarm to watch, and the threshold for that metric. The alarm goes to **ALARM** state when the metric breaches the threshold for a specified number of evaluation periods.

To create an alarm based on a single metric

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. (Optional) If you have enabled cross-account functionality in the CloudWatch console and the current account is a monitoring account, under **Search Metrics** choose a different AWS account that contains the metric that you want the alarm to watch. For more information, see [Cross-account cross-Region CloudWatch console \(p. 284\)](#).
6. Do one of the following:
 - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
 - In the search box, enter the name of a metric, dimension, or resource ID and press Enter. Then choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.
7. Choose the **Graphed metrics** tab.
 - a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
 - b. Under **Period**, choose the evaluation period for the alarm. When evaluating the alarm, each period is aggregated into one data point.

You can also choose whether the y-axis legend appears on the left or right while you're creating the alarm. This preference is used only while you're creating the alarm.

- c. Choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic you have selected.

8. Under **Conditions**, specify the following:

- a. For **Whenever metric is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
- b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the **ALARM** state to trigger the alarm. If the two values here match, you create an alarm that goes to **ALARM** state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm \(p. 131\)](#).

- c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data \(p. 132\)](#).
- d. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples \(p. 136\)](#).

9. Choose **Next**.

10. Under **Notification**, select an SNS topic to notify when the alarm is in **ALARM** state, **OK** state, or **INSUFFICIENT_DATA** state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

11. To have the alarm perform Auto Scaling, EC2, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions only when they go into **ALARM** state. For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Note

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

12. When finished, choose **Next**.

13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.

14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see [Add an alarm widget to a CloudWatch dashboard \(p. 32\)](#).

Creating a CloudWatch alarm based on anomaly detection

You can create an alarm based on CloudWatch anomaly detection, which analyzes past metric data and creates a model of expected values. The expected values take into account the typical hourly, daily, and weekly patterns in the metric.

You set a value for the anomaly detection threshold, and CloudWatch uses this threshold with the model to determine the "normal" range of values for the metric. A higher value for the threshold produces a thicker band of "normal" values.

You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or either above or below the band.

You also can create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands.

Cross-account or cross-Region alarms based on anomaly detection are not supported.

For more information, see [Using CloudWatch anomaly detection \(p. 289\)](#).

Note

If you're already using anomaly detection for visualization purposes on a metric in the Metrics console and you create an anomaly detection alarm on that same metric, then the threshold that you set for the alarm doesn't change the threshold that you already set for visualization. For more information, see [Creating a graph \(p. 85\)](#).

To create an alarm based on anomaly detection

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Choose **Select Metric** and do one of the following:
 - Choose the service namespace that contains the metric that you want. To narrow the choices, continue choosing options as they appear. When a list of metrics appears, select the check box next to the metric that you want.
 - In the search box, enter the name of a metric, dimension, or resource ID and press Enter. Then choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.
6. Choose the **Graphed metrics** tab.
 - a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95 . 45**).
 - b. Under **Period**, choose the evaluation period for the alarm. When evaluating the alarm, each period is aggregated into one data point. For anomaly detection alarms, the value must be one minute or longer.

You can also choose whether the y-axis legend appears on the left or right while you're creating the alarm. This preference is used only while you're creating the alarm.

- c. Choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic you have selected.

7. Under **Conditions**, specify the following:

a. Choose **Anomaly detection**.

If the model for this metric and statistic already exists, CloudWatch displays the anomaly detection band in the sample graph at the top of the screen. If the model does not already exist, the model is generated when you are done creating the alarm. It takes up to 15 minutes for the actual anomaly detection band generated by the model to appear in the graph. Before that, the band you see is an approximation of the anomaly detection band. To see the graph in a longer time frame, choose **Edit** at the top right of the page.

- b. For **Whenever metric is**, specify when to trigger the alarm. For example, when the metric is greater than, lower than, or outside the band (in either direction).
- c. For **Anomaly detection threshold**, choose the number to use for the anomaly detection threshold. A higher number creates a thicker band of "normal" values that is more tolerant of metric changes, and a lower number creates a thinner band that will go to **ALARM** state with smaller metric deviations. The number does not have to be a whole number.
- d. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the **ALARM** state to trigger the alarm. If the two values here match, you create an alarm that goes to **ALARM** state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a number for the first value that is low than the number for the second value. For more information, see [Evaluating an alarm \(p. 131\)](#).

- e. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data \(p. 132\)](#).
- f. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples \(p. 136\)](#).

8. Choose **Next**.

9. Under **Notification**, select an SNS topic to notify when the alarm is in **ALARM** state, **OK** state, or **INSUFFICIENT_DATA** state.

To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

Choose **Remove** if you don't want the alarm to send notifications.

10. You can set up the alarm to perform EC2 actions when it changes state, or to create a Systems Manager OpsItem or incident when it goes into **ALARM** state. To do this, choose the appropriate button and then choose the alarm state and action to perform.

For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Note

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

11. When finished, choose **Next**.
12. Enter a name and description for the alarm. The name must contain only ASCII characters. Then, choose **Next**.
13. Under **Preview and create**, confirm that the information and conditions are what you want, and then choose **Create alarm**.

Modifying an anomaly detection model

After you create an alarm, you can adjust the anomaly detection model. You can exclude certain time periods from being used in the model creation. It is critical that you exclude unusual events such as system outages, deployments, and holidays from the training data. You can also specify whether to adjust the model for Daylight Savings Time changes.

To adjust the anomaly detection model for an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose the name of the alarm. Use the search box to find the alarm if necessary.
4. Choose **View in metrics**.
5. In the lower part of the screen, choose **Edit model**.
6. To exclude a time period from being used to produce the model, choose **Add another time range to exclude from training**. Then select or enter the days and times to exclude from training and choose **Apply**.
7. If the metric is sensitive to Daylight Savings Time changes, select the appropriate time zone in the **Metric timezone** box.
8. Choose **Update**.

Deleting an anomaly detection model

Using anomaly detection for an alarm accrues AWS charges. If you no longer need an anomaly detection model for an alarm, you should delete the alarm and then the model. If you delete the model without deleting the alarm, the alarm automatically recreates the model.

To delete an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose the name of the alarm.
4. Choose **Actions, Delete**.

To delete the anomaly detection model that was used for an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, enter a search term in the search field, such as a metric name or resource name, and press Enter.

For example, if you search for the `CPUUtilization` metric, you see the namespaces and dimensions with this metric.

4. In the results, select the metric that had the anomaly detection model.
5. Choose the **Graphed metrics** tab.
6. In the lower part of the screen, choose **Edit model** and then choose **Delete model**.

Creating a CloudWatch alarm based on a metric math expression

To create an alarm based on a metric math expression, choose one or more CloudWatch metrics to use in the expression. Then, specify the expression, threshold, and evaluation periods.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To create an alarm that's based on a metric math expression

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**, and then perform one of the following actions:
 - Select a namespace from the **AWS namespaces** dropdown or **Custom namespaces** dropdown. After you select a namespace, you continue choosing options until a list of metrics appears, where you select the checkbox next to the correct metric.
 - Use the search box to find a metric, dimension, or resource ID. After you enter the metric, dimension, or resource ID, you continue choosing options until a list of metrics appears, where you select the check box next to the correct metric.
5. (Optional) If your current AWS account is a monitoring account, and you added cross-account functionality to your CloudWatch console, you can use the search box to find an AWS account that contains the metric you want your alarm to monitor. For more information about cross-account functionality, see [Cross-account cross-Region CloudWatch console \(p. 284\)](#).

Note

Currently, you can create cross-account alarms, not cross-Region alarms.

6. (Optional) If you want to add another metric to a metric math expression, you can use the search box to find a specific metric. You can add as many as 10 metrics to a metric math expression.
7. Select the **Graphed metrics** tab. For each of the metrics that you previously added, perform the following actions:
 - a. Under the **Statistic** column, select the dropdown menu. In the dropdown menu, choose one of the predefined statistics or percentiles. Use the search box in the dropdown menu to specify a custom percentile.
 - b. Under the **Period** column, select the dropdown menu. In the dropdown menu, choose one of the predefined evaluation periods.

While you're creating your alarm, you can specify whether the Y-axis legend appears on the left or right side of your graph.

Note

When CloudWatch evaluates alarms, periods are aggregated into single data points.

8. Choose the **Add math** dropdown, and then select **Start with an empty expression** from the list of predefined metric math expressions.

After you choose **Start with an empty expression**, a math expression box appears where you apply or edit math expressions.

9. In the math expression box, enter your math expression, and then choose **Apply**.

After you choose **Apply**, an **ID** column appears next to the **Label** column.

To use a metric or the result of another metric math expression as part of your current math expression's formula, you use the value that's shown under the **ID** column. To change the value of **ID**, you select the pen-and-paper icon next to the current value. The new value must begin with a lowercase letter and can include numbers, letters, and the underscore symbol. Changing the value of **ID** to a more significant name can make your alarm graph easier to understand.

For information about the functions that are available for metric math, see [Metric math syntax and functions \(p. 98\)](#).

10. (Optional) Add more math expressions, using both metrics and the results of other math expressions in the formulas of the new math expressions.
11. When you have the expression to use for the alarm, clear the check boxes to the left of every other expression and every metric on the page. Only the check box next to the expression to use for the alarm should be selected. The expression that you choose for the alarm must produce a single time series and show only one line on the graph. Then choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

12. For **Whenever expression is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
13. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the **ALARM** state to trigger the alarm. If the two values here match, you create an alarm that goes to **ALARM** state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm \(p. 131\)](#).

14. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data \(p. 132\)](#).
15. Choose **Next**.
16. Under **Notification**, select an SNS topic to notify when the alarm is in **ALARM** state, **OK** state, or **INSUFFICIENT_DATA** state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

17. To have the alarm perform Auto Scaling, EC2, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions only when they go into **ALARM** state. For more information about Systems Manager actions, see see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Note

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

18. When finished, choose **Next**.
19. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
20. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see [Add an alarm widget to a CloudWatch dashboard \(p. 32\)](#).

Creating a composite alarm

Composite alarms are alarms that determine their alarm state by watching the alarm states of other alarms.

Using composite alarms can help you reduce alarm noise. If you set up a composite alarm to notify you of state changes, but set up the underlying metric alarms to not send notifications themselves, you will be notified only when the alarm state of the composite alarm changes. For example, you could create metric alarms based on both CPU utilization and disk read operations, and specify for these alarms to never take actions. You could then create a composite alarm that goes into ALARM state and notifies you only when both of those metric alarms are in ALARM state.

In a composite alarm, all underlying alarms must be in the same AWS Region and the same account. Each composite alarm can watch as many as 100 underlying alarms. A single alarm can be watched by up to 150 composite alarms.

Currently, the alarm actions that can be taken by composite alarms are notifying SNS topics, or creating OpsItems in Systems Manager Ops Center or incidents in AWS Systems Manager Incident Manager.

Rule expressions

Each composite alarms includes a *rule expression*, which specifies which other alarms are to be evaluated to determine the composite alarm's state. For each alarm that you reference in the rule expression, you designate a function that specifies whether that alarm needs to be in ALARM state, OK state, or INSUFFICIENT_DATA state. You can use operators (AND, OR and NOT) to combine multiple functions in a single expression. You can use parenthesis to logically group the functions in your expression.

A rule expression can refer to both metric alarms and other composite alarms.

Functions can include the following:

- `ALARM("alarm-name or alarm-ARN")` is TRUE if the named alarm is in ALARM state.
- `OK("alarm-name or alarm-ARN")` is TRUE if the named alarm is in OK state.
- `INSUFFICIENT_DATA("alarm-name or alarm-ARN")` is TRUE if the named alarm is in INSUFFICIENT_DATA state.
- `TRUE` always evaluates to TRUE.
- `FALSE` always evaluates to FALSE.

TRUE and FALSE are useful for testing a complex `AlarmRule` structure, and for testing your alarm actions.

The following are some examples of `AlarmRule`:

- `ALARM(CPUUtilizationTooHigh) AND ALARM(DiskReadOpsTooHigh)` specifies that the composite alarm goes into ALARM state only if both `CPUUtilizationTooHigh` and `DiskReadOpsTooHigh` alarms are in ALARM state.
- `ALARM(CPUUtilizationTooHigh) AND NOT ALARM(DeploymentInProgress)` specifies that the alarm goes to ALARM state if `CPUUtilizationTooHigh` is in ALARM state and `DeploymentInProgress` is not in ALARM state. This example reduces alarm noise during a known deployment window.
- `(ALARM(CPUUtilizationTooHigh) OR ALARM(DiskReadOpsTooHigh)) AND OK(NetworkOutTooHigh)` goes into ALARM state if `CPUUtilizationTooHigh` OR `DiskReadOpsTooHigh` is in ALARM state, and if `NetworkOutTooHigh` is in OK state. This provides another example of using a composite alarm to prevent noise. This rule ensures that you are not notified with an alarm action on high CPU or disk usage if a known network problem is also occurring.

An **AlarmRule** expression can specify as many as 100 "child" alarms. The **AlarmRule** expression can have as many as 500 elements. Elements are child alarms, TRUE or FALSE statements, and parentheses. A pair of parentheses counts as one element.

A rule expression must contain at least one child alarm or at least one TRUE statement or FALSE statement.

To create a composite alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. In the list of alarms, select the check boxes next to each of the existing alarms that you want to reference in your new composite alarm. Then choose **Create composite alarm**.
4. In the **Conditions** box, specify the rule expression that the composite alarm will use. Initially, the alarms you selected are listed, joined by the OR logical operator. Each of these alarms has the ALARM state specified.

You can modify the alarm conditions for the composite alarm that you are creating:

- a. For each underlying alarm listed, you can change the required state from ALARM to OK or INSUFFICIENT_DATA.
- b. You can change each OR operator to AND or NOT. You can also add parentheses to group the logical operators.
- c. You can add more alarms to the composite alarm conditions. You can also delete alarms currently listed in the **Conditions** box.

For example, you could specify the following conditions to create a composite alarm that goes into ALARM state if **CPUUtilizationTooHigh** OR **DiskReadOpsTooHigh** is in ALARM state, at the same time that **NetworkOutTooHigh** is in OK state.

```
(ALARM("CPUUtilizationTooHigh") OR  
ALARM("DiskReadOpsTooHigh")) AND  
OK("NetworkOutTooHigh")
```

5. When you are satisfied with the alarm conditions, choose **Next**.
6. Under **Notification**, select an SNS topic to notify when the composite alarm changes state.

To have the alarm not send notifications, choose **Remove**.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm perform an SSM action when it goes into ALARM state, choose **Add Systems Manager action**.

For more information about Systems Manager actions, see see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Note

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

7. When finished, choose **Next**.
8. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.

9. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Note

It is possible to create a loop or cycle of composite alarms, where composite alarm A depends on composite alarm B, and composite alarm B also depends on composite alarm A. In this scenario, you can't delete any composite alarm that is part of the cycle because there is always still a composite alarm that depends on that alarm that you want to delete.

To get out of such a situation, you must break the cycle by changing the rule of one of these composite alarms to remove a dependency that creates the cycle. The simplest change to make to break a cycle is to change the `AlarmRule` of one of the alarms to `False`.

Additionally, the evaluation of composite alarms stops if CloudWatch detects a cycle in the evaluation path.

Editing or deleting a CloudWatch alarm

You can edit or delete an existing alarm.

To edit an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose the name of the alarm.
4. Choose **Edit**.

The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.

5. To change the metric, choose **Edit**, choose the **All metrics** tab, and do one of the following:
 - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
 - In the search box, enter the name of a metric, dimension, or resource ID and press Enter. Then choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.

Choose **Select metric**.

6. To change other aspects of the alarm, choose the appropriate options. To change how many data points must be breaching for the alarm to go into **ALARM** state or to change how missing data is treated, choose **Additional configuration**.
7. Choose **Next**.
8. Under **Notification**, **Auto Scaling action**, and **EC2 action**, optionally edit the actions taken when the alarm is triggered. Then choose **Next**.
9. Optionally change the alarm description.

You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action, Copy**.

10. Choose **Next**.
11. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Update alarm**.

To update an email notification list that was created using the Amazon SNS console

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose **Topics** and then select the ARN for your notification list (topic).
3. Do one of the following:
 - To add an email address, choose **Create subscription**. For **Protocol**, choose **Email**. For **Endpoint**, enter the email address of the new recipient. Choose **Create subscription**.
 - To remove an email address, choose the **Subscription ID**. Choose **Other subscription actions**, **Delete subscriptions**.
4. Choose **Publish to topic**.

To delete an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**.
3. Select the check box to the left of the name of the alarm, and choose **Actions**, **Delete**.
4. Choose **Delete**.

Creating a CPU usage alarm

You can create an CloudWatch alarm that sends a notification using Amazon SNS when the alarm changes state from **OK** to **ALARM**.

The alarm changes to the **ALARM** state when the average CPU use of an EC2 instance exceeds a specified threshold for consecutive specified periods.

Setting up a CPU usage alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. In the **All metrics** tab, choose **EC2 metrics**.
6. Choose a metric category (for example, **Per-Instance Metrics**).
7. Find the row with the instance that you want listed in the **InstanceId** column and **CPUUtilization** in the **Metric Name** column. Select the check box next to this row, and choose **Select metric**.
8. Under **Specify metric and conditions**, for **Statistic** choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **p95 . 45**).
9. Choose a period (for example, **5 minutes**).
10. Under **Conditions**, specify the following:
 - a. For **Threshold type**, choose **Static**.

- b. For **Whenever CPUUtilization is**, specify **Greater**. Under **than...**, specify the threshold that is to trigger the alarm to go to ALARM state if the CPU utilization exceeds this percentage. For example, 70.
- c. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm \(p. 131\)](#).

- d. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data \(p. 132\)](#).
- e. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples \(p. 136\)](#).

11. Choose **Next**.

12. Under **Notification**, choose **In alarm** and select an SNS topic to notify when the alarm is in ALARM state

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

13. When finished, choose **Next**.

14. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.

15. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Setting up a CPU usage alarm using the AWS CLI

Use these steps to use the AWS CLI to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Set up an SNS topic. For more information, see [Setting up Amazon SNS notifications \(p. 137\)](#).
2. Create an alarm using the `put-metric-alarm` command as follows.

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm when CPU exceeds 70%" --metric-name CPUUtilization --namespace AWS/EC2 --statistic Average --period 300 --threshold 70 --comparison-operator GreaterThanThreshold --dimensions Name=InstanceId,Value=i-12345678 --evaluation-periods 2 --alarm-actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Percent
```

3. Test the alarm by forcing an alarm state change using the `set-alarm-state` command.

- a. Change the alarm state from **INSUFFICIENT_DATA** to **OK**.

```
aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason "initializing" --state-value OK
```

- b. Change the alarm state from OK to ALARM.

```
aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason "initializing"  
--state-value ALARM
```

- c. Check that you have received a notification about the alarm.

Creating a load balancer latency alarm that sends email

You can set up an Amazon SNS notification and configure an alarm that monitors latency exceeding 100 ms for your Classic Load Balancer.

Setting up a latency alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose **Create alarm**.
4. Under **CloudWatch Metrics by Category**, choose the **ELB Metrics** category.
5. Select the row with the Classic Load Balancer and the **Latency** metric.
6. For the statistic, choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **p95 .45**).
7. For the period, choose **1 Minute**.
8. Choose **Next**.
9. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **myHighCpuAlarm**) and a description of the alarm (for example, **Alarm when Latency exceeds 100s**). Alarm names must contain only ASCII characters.
10. Under **Whenever, for is**, choose **>** and enter **0 . 1**. For **for**, enter **3**.
11. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.

For **Percentiles with low samples**, choose **ignore (maintain the alarm state)** so that the alarm evaluates only situations with adequate numbers of data samples.

12. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **myHighCpuAlarm**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent.

13. Choose **Create Alarm**.

Setting up a latency alarm using the AWS CLI

Use these steps to use the AWS CLI to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Set up an SNS topic. For more information, see [Setting up Amazon SNS notifications \(p. 137\)](#).
2. Create the alarm using the [put-metric-alarm](#) command as follows:

```
aws cloudwatch put-metric-alarm --alarm-name lb-mon --alarm-description "Alarm when Latency exceeds 100s" --metric-name Latency --namespace AWS/ELB --statistic Average --period 60 --threshold 100 --comparison-operator GreaterThanThreshold --dimensions Name=LoadBalancerName,Value=my-server --evaluation-periods 3 --alarm-actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Seconds
```

3. Test the alarm by forcing an alarm state change using the [set-alarm-state](#) command.

- a. Change the alarm state from INSUFFICIENT_DATA to OK.

```
aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason "initializing" --state-value OK
```

- b. Change the alarm state from OK to ALARM.

```
aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason "initializing" --state-value ALARM
```

- c. Check that you have received an email notification about the alarm.

Creating a storage throughput alarm that sends email

You can set up an SNS notification and configure an alarm that is triggered when Amazon EBS exceeds 100 MB throughput.

Setting up a storage throughput alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose **Create alarm**.
4. Under **EBS Metrics**, choose a metric category.
5. Select the row with the volume and the **VolumeWriteBytes** metric.
6. For the statistic, choose **Average**. For the period, choose **5 Minutes**. Choose **Next**.

7. Under **Alarm Threshold**, enter a unique name for the alarm (for example, `myHighWriteAlarm`) and a description of the alarm (for example, `VolumeWriteBytes exceeds 100,000 KiB/s`). Alarm names must contain only ASCII characters.
8. Under **Whenever**, for **is**, choose `>` and enter `100000`. For **for**, enter `15` consecutive periods.
A graphical representation of the threshold is shown under **Alarm Preview**.
9. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.
10. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create one.

To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, `myHighCpuAlarm`), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

11. Choose **Create Alarm**.

Setting up a storage throughput alarm using the AWS CLI

Use these steps to use the AWS CLI to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Create an SNS topic. For more information, see [Setting up Amazon SNS notifications \(p. 137\)](#).
2. Create the alarm.

```
aws cloudwatch put-metric-alarm --alarm-name ebs-mon --alarm-description "Alarm when EBS volume exceeds 100MB throughput" --metric-name VolumeReadBytes --namespace AWS/EBS --statistic Average --period 300 --threshold 100000000 --comparison-operator GreaterThanThreshold --dimensions Name=VolumeId,Value=my-volume-id --evaluation-periods 3 --alarm-actions arn:aws:sns:us-east-1:111122223333:my-alarm-topic --insufficient-data-actions arn:aws:sns:us-east-1:111122223333:my-insufficient-data-topic
```

3. Test the alarm by forcing an alarm state change using the `set-alarm-state` command.

- a. Change the alarm state from `INSUFFICIENT_DATA` to `OK`.

```
aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason "initializing" --state-value OK
```

- b. Change the alarm state from `OK` to `ALARM`.

```
aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason "initializing" --state-value ALARM
```

- c. Change the alarm state from `ALARM` to `INSUFFICIENT_DATA`.

```
aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason "initializing" --state-value INSUFFICIENT_DATA
```

- d. Check that you have received an email notification about the alarm.

Create alarms to stop, terminate, reboot, or recover an EC2 instance

Using Amazon CloudWatch alarm actions, you can create alarms that automatically stop, terminate, reboot, or recover your EC2 instances. You can use the stop or terminate actions to help you save money when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances or recover them onto new hardware if a system impairment occurs.

There are a number of scenarios in which you might want to automatically stop or terminate your instance. For example, you might have instances dedicated to batch payroll processing jobs or scientific computing tasks that run for a period of time and then complete their work. Rather than letting those instances sit idle (and accrue charges), you can stop or terminate them, which helps you to save money. The main difference between using the stop and the terminate alarm actions is that you can easily restart a stopped instance if you need to run it again later. You can also keep the same instance ID and root volume. However, you cannot restart a terminated instance. Instead, you must launch a new instance.

You can add the stop, terminate, or reboot, actions to any alarm that is set on an Amazon EC2 per-instance metric, including basic and detailed monitoring metrics provided by Amazon CloudWatch (in the AWS/EC2 namespace), in addition to any custom metrics that include the "InstanceId=" dimension, as long as the InstanceId value refers to a valid running Amazon EC2 instance. You can also add the recover action to alarms that is set on any Amazon EC2 per-instance metric except for StatusCheckFailed_Instance.

To set up a CloudWatch alarm action that can reboot, stop, or terminate an instance, you must use a service-linked IAM role, `AWSServiceRoleForCloudWatchEvents`. The `AWSServiceRoleForCloudWatchEvents` IAM role enables AWS to perform alarm actions on your behalf.

To create the service-linked role for CloudWatch Events, use the following command:

```
aws iam create-service-linked-role --aws-service-name events.amazonaws.com
```

Console support

You can create alarms using the CloudWatch console or the Amazon EC2 console. The procedures in this documentation use the CloudWatch console. For procedures that use the Amazon EC2 console, see [Create Alarms That Stop, Terminate, Reboot, or Recover an Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

Permissions

If you are using an AWS Identity and Access Management (IAM) account to create or modify an alarm that performs EC2 actions or Systems Manager OpsItem actions, you must have the `iam:CreateServiceLinkedRole` permission.

Contents

- [Adding stop actions to Amazon CloudWatch alarms \(p. 155\)](#)
- [Adding terminate actions to Amazon CloudWatch alarms \(p. 156\)](#)
- [Adding reboot actions to Amazon CloudWatch alarms \(p. 156\)](#)
- [Adding recover actions to Amazon CloudWatch alarms \(p. 157\)](#)
- [Viewing the history of triggered alarms and actions \(p. 159\)](#)

Adding stop actions to Amazon CloudWatch alarms

You can create an alarm that stops an Amazon EC2 instance when a certain threshold has been met. For example, you may run development or test instances and occasionally forget to shut them off. You can create an alarm that is triggered when the average CPU utilization percentage has been lower than 10 percent for 24 hours, signaling that it is idle and no longer in use. You can adjust the threshold, duration, and period to suit your needs, plus you can add an SNS notification, so that you will receive an email when the alarm is triggered.

Amazon EC2 instances that use an Amazon Elastic Block Store volume as the root device can be stopped or terminated, whereas instances that use the instance store as the root device can only be terminated.

To create an alarm to stop an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. For **AWS namespaces**, choose **EC2**.
6. Do the following:
 - a. Choose **Per-Instance Metrics**.
 - b. Select the check box in the row with the correct instance and the **CPUUtilization** metric.
 - c. Choose **Graphed metrics**.
 - d. For the statistic, choose **Average**.
 - e. Choose a period (for example, **1 Hour**).
 - f. Choose **Select metric**.
7. For the **Define Alarm** step, do the following:
 - a. Under **Conditions**, choose **Static**.
 - b. Under **Whenever CPUUtilization is**, choose **Lower**.
 - c. For **than**, type **10**.
 - d. Choose **Next**.
 - e. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, **Stop_EC2_Instance**). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

- f. Choose **Add EC2 Action**.
- g. For **Alarm state trigger**, choose **In alarm**. For **Take the following action**, choose **Stop this instance**.
- h. Choose **Next**.
- i. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
- j. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Adding terminate actions to Amazon CloudWatch alarms

You can create an alarm that terminates an EC2 instance automatically when a certain threshold has been met (as long as termination protection is not enabled for the instance). For example, you might want to terminate an instance when it has completed its work, and you don't need the instance again. If you might want to use the instance later, you should stop the instance instead of terminating it. For information about enabling and disabling termination protection for an instance, see [Enabling Termination Protection for an Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

To create an alarm to terminate an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, Create Alarm**.
3. For the **Select Metric** step, do the following:
 - a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
 - b. Select the row with the instance and the **CPUUtilization** metric.
 - c. For the statistic, choose **Average**.
 - d. Choose a period (for example, **1 Hour**).
 - e. Choose **Next**.
4. For the **Define Alarm** step, do the following:
 - a. Under **Alarm Threshold**, type a unique name for the alarm (for example, **Terminate EC2 instance**) and a description of the alarm (for example, **Terminate EC2 instance when CPU is idle for too long**). Alarm names must contain only ASCII characters.
 - b. Under **Whenever, for is**, choose **<** and type **10**. For **for**, type **24** consecutive periods.

A graphical representation of the threshold is shown under **Alarm Preview**.
 - c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.
5. To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, **Terminate_EC2_Instance**). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.
6. Choose **EC2 Action**.
7. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Terminate this instance**.
8. Choose **Create Alarm**.

Adding reboot actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically reboots the instance. The reboot alarm action is recommended for Instance Health Check failures (as opposed to the recover alarm action, which is suited for System Health Check failures). An instance reboot is equivalent to an operating system reboot. In most cases, it takes only a few minutes to reboot your instance. When you reboot an instance, it remains on the same physical host, so your instance keeps its public DNS name, private IP address, and any data on its instance store volumes.

Rebooting an instance doesn't start a new instance billing hour, unlike stopping and restarting your instance. For more information about rebooting an instance, see [Reboot Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

Important

To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set reboot alarms to three evaluation periods of one minute each.

To create an alarm to reboot an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, **Create Alarm**.
3. For the **Select Metric** step, do the following:
 - a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
 - b. Select the row with the instance and the **StatusCheckFailed_Instance** metric.
 - c. For the statistic, choose **Minimum**.
 - d. Choose a period (for example, **1 Minute**) and choose **Next**.
4. For the **Define Alarm** step, do the following:
 - a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Reboot EC2 instance) and a description of the alarm (for example, Reboot EC2 instance when health checks fail). Alarm names must contain only ASCII characters.
 - b. Under **Whenever**, for **is**, choose **>** and type **0**. For **for**, type **3** consecutive periods.
A graphical representation of the threshold is shown under **Alarm Preview**.
 - c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one. To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Reboot_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.
 - d. Choose **EC2 Action**.
 - e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Reboot this instance**.
 - f. Choose **Create Alarm**.

Adding recover actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically recovers the instance if it becomes impaired due to an underlying hardware failure or a problem that requires AWS involvement to repair. Terminated instances cannot be recovered. A recovered instance is identical to the original instance, including the instance ID, private IP addresses, Elastic IP addresses, and all instance metadata.

When the `statusCheckFailed_System` alarm is triggered, and the recover action is initiated, you will be notified by the Amazon SNS topic that you chose when you created the alarm and associated the recover action. During instance recovery, the instance is migrated during an instance reboot, and any data that is in-memory is lost. When the process is complete, information is published to the SNS topic you've configured for the alarm. Anyone who is subscribed to this SNS topic will receive an email

notification that includes the status of the recovery attempt and any further instructions. You will notice an instance reboot on the recovered instance.

The recover action can be used only with `StatusCheckFailed_System`, not with `StatusCheckFailed_Instance`.

Examples of problems that cause system status checks to fail include:

- Loss of network connectivity
- Loss of system power
- Software issues on the physical host
- Hardware issues on the physical host that impact network reachability

The recover action is supported only on:

- The A1, C3, C4, C5, C5a, C5n, C6g, Inf1, M3, M4, M5, M5a, M5n, M5zn, M6g, P3, P4, R3, R4, R5, R5a, R5b, R5n, T2, T3, T3a, T4g, X1, X1e, X2idn, and X2iedn instance types
- Instances in a VPC
- Instances with default or dedicated instance tenancy
- Instances that use Amazon EBS volumes only (do not configure instance store volumes)

If your instance has a public IPv4 address, it retains the public IP address after recovery.

Important

If the instance has store volumes attached, the data is lost during recovery.

Important

To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set recover alarms to two evaluation periods of one minute each and reboot alarms to three evaluation periods of one minute each.

To create an alarm to recover an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, Create Alarm**.
3. For the **Select Metric** step, do the following:
 - a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
 - b. Select the row with the instance and the `StatusCheckFailed_System` metric.
 - c. For the statistic, choose **Minimum**.
 - d. Choose a period (for example, **1 Minute**).
4. For the **Define Alarm** step, do the following:
 - a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Recover EC2 instance) and a description of the alarm (for example, Recover EC2 instance when health checks fail). Alarm names must contain only ASCII characters.
 - b. Under **Whenever, for is**, choose **>** and type **0**. For **for**, type **2** consecutive periods.
 - c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Recover_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the **ALARM** state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

- d. Choose **EC2 Action**.
- e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Recover this instance**.
- f. Choose **Create Alarm**.

Viewing the history of triggered alarms and actions

You can view alarm and action history in the Amazon CloudWatch console. Amazon CloudWatch keeps the last two weeks' worth of alarm and action history.

To view the history of triggered alarms and actions

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms** and select an alarm.
3. To view the most recent state transition along with the time and metric values, choose **Details**.
4. To view the most recent history entries, choose **History**.

Creating a billing alarm to monitor your estimated AWS charges

You can monitor your estimated AWS charges by using Amazon CloudWatch. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

Billing metric data is stored in the US East (N. Virginia) Region and represents worldwide charges. This data includes the estimated charges for every service in AWS that you use, in addition to the estimated overall total of your AWS charges.

The alarm triggers when your account billing exceeds the threshold you specify. It triggers only when the current billing exceeds the threshold. It doesn't use projections based on your usage so far in the month.

If you create a billing alarm at a time when your charges have already exceeded the threshold, the alarm goes to the **ALARM** state immediately.

Tasks

- [Enabling billing alerts \(p. 159\)](#)
- [Creating a billing alarm \(p. 160\)](#)
- [Deleting a billing alarm \(p. 161\)](#)

Enabling billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated AWS charges and create an alarm using billing metric data. After you enable billing alerts, you can't disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using account root user credentials or as an IAM user that has been given permission to view billing information.
- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see [AWS Partner Network](#).

To enable the monitoring of estimated charges

1. Open the AWS Billing console at <https://console.aws.amazon.com/billing/>.
2. In the navigation pane, choose **Billing Preferences**.
3. Choose **Receive Billing Alerts**.
4. Choose **Save preferences**.

Creating a billing alarm

Important

Before you can create a billing alarm, you must enable billing alerts in your account, or in the management/payer account if you are using consolidated billing. For more information, see [Enabling billing alerts \(p. 159\)](#).

In this procedure, you create an alarm that sends a notification when your estimated charges for AWS exceed a specified threshold. This procedure uses the advanced options. For more information about using the simple options, see [Create a Billing Alarm \(p. 841\)](#) in *Monitor Your Estimated Charges Using CloudWatch*.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges.
3. In the navigation pane, choose **Alarms, All alarms**.
4. Choose **Create alarm**.
5. Choose **Select metric**. In the **All metrics** tab, choose **Billing, Total Estimated Charge**.

If you don't see **Billing** or the **Total Estimated Charge** metric, you may need to enable billing alerts. For more information, see [Enabling billing alerts \(p. 159\)](#).

6. Select the check box next to **EstimatedCharges**, and choose **Select metric**.
7. Under **Conditions**, choose **Static**.
8. For **Whenever EstimatedCharges is**, choose **Greater**.
9. For **than**, enter the monthly amount (for example, **200**) that must be exceeded to trigger the alarm.

Note

The preview graph displays your current charges for the month.

10. Choose **Next**.
11. For **Select an SNS topic**, select an SNS topic to notify when the alarm is in **ALARM** state, or create a new topic to be notified.
To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.
12. When finished, choose **Next**.
13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Deleting a billing alarm

You can delete your billing alarm when you no longer need it.

To delete a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms, All alarms**.
4. Select the check box next to the alarm and choose **Actions, Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Hiding Amazon EC2 Auto Scaling alarms

When you view your alarms in the AWS Management Console, you can hide the alarms related to Amazon EC2 Auto Scaling. This feature is available only in the AWS Management Console.

To temporarily hide Amazon EC2 Auto Scaling alarms

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**, and select **Hide Auto Scaling alarms**.

Using synthetic monitoring

You can use Amazon CloudWatch Synthetics to create *canaries*, configurable scripts that run on a schedule, to monitor your endpoints and APIs. Canaries follow the same routes and perform the same actions as a customer, which makes it possible for you to continually verify your customer experience even when you don't have any customer traffic on your applications. By using canaries, you can discover issues before your customers do.

Canaries are scripts written in Node.js or Python. They create Lambda functions in your account that use Node.js or Python as a framework. Canaries work over both HTTP and HTTPS protocols.

Canaries offer programmatic access to a headless Google Chrome Browser via Puppeteer or Selenium Webdriver. For more information about Puppeteer, see [Puppeteer](#). For more information about Selenium, see [www.selenium.dev/](#).

Canaries check the availability and latency of your endpoints and can store load time data and screenshots of the UI. They monitor your REST APIs, URLs, and website content, and they can check for unauthorized changes from phishing, code injection and cross-site scripting.

For a video demonstration of canaries, see the following:

- [Introduction to Amazon CloudWatch Synthetics](#)
- [Amazon CloudWatch Synthetics Demo](#)
- [Create Canaries Using Amazon CloudWatch Synthetics](#)
- [Visual Monitoring with Amazon CloudWatch Synthetics](#)

You can run a canary once or on a regular schedule. Canaries can run as often as once per minute. You can use both cron and rate expressions to schedule canaries.

For information about security issues to consider before you create and run canaries, see [Security considerations for Synthetics canaries \(p. 955\)](#).

By default, canaries create several CloudWatch metrics in the `CloudWatchSynthetics` namespace. These metrics have `CanaryName` as a dimension. Canaries that use the `executeStep()` or `executeHttpStep()` function from the function library also have `StepName` as a dimension. For more information about the canary function library, see [Library functions available for canary scripts \(p. 200\)](#).

CloudWatch Synthetics integrates well with CloudWatch ServiceLens, which uses CloudWatch with AWS X-Ray to provide an end-to-end view of your services to help you more efficiently pinpoint performance bottlenecks and identify impacted users. Canaries that you create with CloudWatch Synthetics appear on the ServiceLens service map. For more information about ServiceLens, see [Using ServiceLens to monitor the health of your applications \(p. 268\)](#).

CloudWatch Synthetics is currently available in all commercial AWS Regions and the GovCloud Regions.

Note

In Asia Pacific (Osaka), AWS PrivateLink is not supported. In Asia Pacific (Jakarta), AWS PrivateLink and X-Ray are not supported.

Topics

- [Required roles and permissions for CloudWatch canaries \(p. 163\)](#)
- [Creating a canary \(p. 173\)](#)
- [Troubleshooting a failed canary \(p. 232\)](#)
- [Sample code for canary scripts \(p. 235\)](#)

- [Canaries and X-Ray tracing \(p. 239\)](#)
- [Running a canary on a VPC \(p. 240\)](#)
- [Encrypting canary artifacts \(p. 241\)](#)
- [Viewing canary statistics and details \(p. 243\)](#)
- [CloudWatch metrics published by canaries \(p. 244\)](#)
- [Editing or deleting a canary \(p. 246\)](#)
- [Monitoring canary events with Amazon EventBridge \(p. 247\)](#)

Required roles and permissions for CloudWatch canaries

Both the users who create and manage canaries, and the canaries themselves, must have certain permissions.

Required roles and permissions for users who manage CloudWatch canaries

To view canary details and the results of canary runs, you must be signed in as an IAM user who has either the `CloudWatchSyntheticsFullAccess` or the `CloudWatchSyntheticsReadOnlyAccess` attached. To read all Synthetics data in the console, you also need the `AmazonS3ReadOnlyAccess` and `CloudWatchReadOnlyAccess` policies. To view the source code used by canaries, you also need the `AWSLambda_ReadOnlyAccess` policy.

To create canaries, you must be signed in as an IAM user who has the `CloudWatchSyntheticsFullAccess` policy or a similar set of permissions. To create IAM roles for the canaries, you also need the following inline policy statement:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:CreateRole",  
                "iam:CreatePolicy",  
                "iam:AttachRolePolicy"  
            ],  
            "Resource": [  
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",  
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"  
            ]  
        }  
    ]  
}
```

Important

Granting a user the `iam:CreateRole`, `iam:CreatePolicy`, and `iam:AttachRolePolicy` permissions gives that user full administrative access to your AWS account. For example, a user with these permissions can create a policy that has full permissions for all resources and can attach that policy to any role. Be very careful about who you grant these permissions to.

For information about attaching policies and granting permissions to users, see [Changing Permissions for an IAM User](#) and [To embed an inline policy for a user or role](#).

Required roles and permissions for canaries

Each canary must be associated with an IAM role that has certain permissions attached. When you create a canary using the CloudWatch console, you can choose for CloudWatch Synthetics to create an IAM role for the canary. If you do, the role will have the permissions needed.

If you want to create the IAM role yourself, or create an IAM role that you can use when using the AWS CLI or APIs to create a canary, the role must contain the permissions listed in this section.

All IAM roles for canaries must include the following trust policy statement.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "lambda.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Additionally, the canary's IAM role needs one of the following statements.

Basic canary that doesn't use AWS KMS or need Amazon VPC access

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::path/to/your/s3/bucket/canary/results/folder"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetBucketLocation"  
            ],  
            "Resource": [  
                "arn:aws:s3:::name/of/the/s3/bucket/that/contains/canary/results"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogStream",  
                "logs:PutLogEvents",  
                "logs>CreateLogGroup"  
            ],  
            "Resource": [  
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/lambda/  
cwsyn-canary_name-*"  
            ]  
        },  
    ]  
}
```

```
{
    "Effect": "Allow",
    "Action": [
        "s3>ListAllMyBuckets",
        "xray:PutTraceSegments"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
}
]
```

Canary that uses AWS KMS to encrypt canary artifacts but does not need Amazon VPC access

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

```

        "*"
    ],
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource":
"arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": [
                "S3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
            ]
        }
    }
}
]
}

```

Canary that does not use AWS KMS but does need Amazon VPC access

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/lambda/
cwsyn-canary_name-*"
            ]
        }
    ]
}

```

```

        ],
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3>ListAllMyBuckets",
            "xray:PutTraceSegments"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "CloudWatchSynthetics"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2>CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2>DeleteNetworkInterface"
        ],
        "Resource": [
            "*"
        ]
    }
]
}

```

Canary that uses AWS KMS to encrypt canary artifacts and also needs Amazon VPC access

If you update a non-VPC canary to start using a VPC, you'll need to update the canary's role to include the network interface permissions listed in the following policy.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",

```

```

    "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs>CreateLogGroup"
    ],
    "Resource": [
        "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/lambda/
cwsyn-canary_name-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3>ListAllMyBuckets",
        "xray:PutTraceSegments"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "ec2>CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2>DeleteNetworkInterface"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource":
"arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": [
                "S3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
            ]
        }
    }
}
]
}

```

AWS managed policies for CloudWatch Synthetics

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](#) 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 change the permissions in an AWS managed policy. This type of update affects all identities (users, groups, and roles) where the policy is attached.

CloudWatch Synthetics updates to AWS managed policies

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

| Change | Description | Date |
|--|--|----------------|
| Redundant actions removed from CloudWatchSyntheticsFullAccess | CloudWatch Synthetics removed the <code>s3:PutBucketEncryption</code> and <code>lambda:GetLayerVersionByArn</code> actions from CloudWatchSyntheticsFullAccess policy because those actions were redundant with other permissions in the policy. The removed actions did not provide any permissions, and there's no net change to the permissions granted by the policy. | March 12, 2021 |
| CloudWatch Synthetics started tracking changes | CloudWatch Synthetics started tracking changes for its AWS managed policies. | March 10, 2021 |

CloudWatchSyntheticsFullAccess

Here are the contents of the **CloudWatchSyntheticsFullAccess** policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::cw-syn-results-*"
            ]
        }
    ]
}
```

```
{
    "Effect": "Allow",
    "Action": [
        "iam>ListRoles",
        "s3>ListAllMyBuckets",
        "s3>GetBucketLocation",
        "xray>GetTraceSummaries",
        "xray>BatchGetTraces",
        "apigateway:GET"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3>GetObject",
        "s3>ListBucket"
    ],
    "Resource": "arn:aws:s3:::cw-syn-*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3>GetObjectVersion"
    ],
    "Resource": "arn:aws:s3:::aws-synthetics-library-*"
},
{
    "Effect": "Allow",
    "Action": [
        "iam>PassRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
    ],
    "Condition": {
        "StringEquals": {
            "iam>PassedToService": [
                "lambda.amazonaws.com",
                "synthetics.amazonaws.com"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam>GetRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch>GetMetricData",
        "cloudwatch>GetMetricStatistics"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch>PutMetricAlarm",
        "cloudwatch>DeleteAlarms"
    ]
}
```

```
        ],
        "Resource":[
            "arn:aws:cloudwatch:*::alarm:Synthetics-*"
        ]
    },
    {
        "Effect":"Allow",
        "Action":[
            "cloudwatch:DescribeAlarms"
        ],
        "Resource":[
            "arn:aws:cloudwatch:*::alarm:/*"
        ]
    },
    {
        "Effect":"Allow",
        "Action":[
            "lambda>CreateFunction",
            "lambda>AddPermission",
            "lambda>PublishVersion",
            "lambda>UpdateFunctionConfiguration",
            "lambda>GetFunctionConfiguration"
        ],
        "Resource":[
            "arn:aws:lambda:*::function:cwsyn-*"
        ]
    },
    {
        "Effect":"Allow",
        "Action":[
            "lambda>GetLayerVersion",
            "lambda>PublishLayerVersion"
        ],
        "Resource":[
            "arn:aws:lambda:*::layer:cwsyn-*",
            "arn:aws:lambda:*::layer:Synthetics:/*"
        ]
    },
    {
        "Effect":"Allow",
        "Action":[
            "ec2>DescribeVpcs",
            "ec2>DescribeSubnets",
            "ec2>DescribeSecurityGroups"
        ],
        "Resource":[
            "*"
        ]
    },
    {
        "Effect":"Allow",
        "Action":[
            "sns>ListTopics"
        ],
        "Resource":[
            "*"
        ]
    },
    {
        "Effect":"Allow",
        "Action":[
            "sns>CreateTopic",
            "sns>Subscribe",
            "sns>ListSubscriptionsByTopic"
        ],
        "Resource":[

```

```
        "arn:*:sns:*:*:Synthetics-*"
    }
}
}
```

CloudWatchSyntheticsReadOnlyAccess

Here are the contents of the `CloudWatchSyntheticsReadOnlyAccess` policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "synthetics:Describe*",
        "synthetics:Get*",
        "synthetics>List*"
      ],
      "Resource": "*"
    }
  ]
}
```

Limiting a user to viewing specific canaries

You can limit a user's ability to view information about canaries, so that they can only see information about the canaries you specify. To do this, use an IAM policy with a `Condition` statement similar to the following, and attach this policy to an IAM user or IAM role.

The following example limits the user to only viewing information about `name-of-allowed-canary-1` and `name-of-allowed-canary-2`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "synthetics:DescribeCanaries",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "synthetics:Names": [
            "name-of-allowed-canary-1",
            "name-of-allowed-canary-2"
          ]
        }
      }
    }
  ]
}
```

CloudWatch Synthetics supports listing as many as five items in the `synthetics:Names` array.

You can also create a policy that uses a `*` as a wildcard in canary names that are to be allowed, as in the following example:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": "synthetics:DescribeCanaries",  
            "Resource": "*",  
            "Condition": {  
                "ForAnyValue:StringLike": {  
                    "synthetics:Names": [  
                        "my-team-canary-*"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

Any user signed in with one of these policies attached can't use the CloudWatch console to view any canary information. They can view canary information only for the canaries authorized by the policy and only by using the [DescribeCanaries](#) API or the [describe-canaries](#) AWS CLI command.

Creating a canary

Important

Ensure that you use Synthetics canaries to monitor only endpoints and APIs where you have ownership or permissions. Depending on the canary frequency settings, these endpoints might experience increased traffic.

When you use the CloudWatch console to create a canary, you can use a blueprint provided by CloudWatch to create your canary or you can write your own script. For more information, see [Using canary blueprints \(p. 175\)](#).

You can also create a canary using AWS CloudFormation if you are using your own script for the canary. For more information, see [AWS::Synthetics::Canary](#) in the *AWS CloudFormation User Guide*.

If you are writing your own script, you can use several functions that CloudWatch Synthetics has built into a library. For more information, see [Synthetics runtime versions \(p. 181\)](#).

To create a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Canaries**.
3. Choose **Create Canary**.
4. Choose one of the following:
 - To base your canary on a blueprint script, choose **Use a blueprint**, and then choose the type of canary you want to create. For more information about what each type of blueprint does, see [Using canary blueprints \(p. 175\)](#).
 - To upload your own Node.js script to create a custom canary, choose **Upload a script**.

You can then drag your script into the **Script** area or choose **Browse files** to navigate to the script in your file system.

- To import your script from an S3 bucket, choose **Import from S3**. Under **Source location**, enter the complete path to your canary or choose **Browse S3**.

You must have `s3:GetObject` and `s3:GetObjectVersion` permissions for the S3 bucket that you use. The bucket must be in the same AWS Region where you are creating the canary.

5. Under **Name**, enter a name for your canary. The name is used on many pages, so we recommend that you give it a descriptive name that distinguishes it from other canaries.
6. Under **Application or endpoint URL**, enter the URL that you want the canary to test. This URL must include the protocol (such as `https://`).

If you want the canary to test an endpoint on a VPC, you must also enter information about your VPC later in this procedure.

7. If you are using your own script for the canary, under **Lambda handler**, enter the entry point where you want the canary to start. If you use a runtime earlier than `syn-nodejs-puppeteer-3.4` or `syn-python-selenium-1.1`, the string that you enter must end with `.handler`. If you use `syn-nodejs-puppeteer-3.4` or `syn-python-selenium-1.1` or a later runtime, this restriction does not apply.
8. If you are using environment variables in your script, choose **Environment variables** and then specify a value for each environment variable defined in your script. For more information, see [Environment variables \(p. 195\)](#).
9. Under **Schedule**, choose whether to run this canary just once, run it continuously using a rate expression, or schedule it using a cron expression.
 - When you use the CloudWatch console to create a canary that runs continuously, you can choose a rate anywhere between once a minute and once an hour.
 - For more information about writing a cron expression for canary scheduling, see [Scheduling canary runs using cron \(p. 230\)](#).
10. (Optional) To set a timeout value for the canary, choose **Additional configuration** and then specify the timeout value. Make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.
11. Under **Data retention**, specify how long to retain information about both failed and successful canary runs. The range is 1-455 days.
12. Under **Data Storage**, select the S3 bucket to use to store the data from the canary runs. The bucket name can't contain a period (.). If you leave this blank, a default S3 bucket is used or created.

If you are using the `syn-nodejs-puppeteer-3.0` or later runtime, when you enter the URL for the bucket in the text box, you can specify a bucket in the current Region or in a different Region. If you are using an earlier runtime version, the bucket must be in the current Region.

13. (Optional) By default, canaries store their artifacts on Amazon S3, and the artifacts are encrypted at rest using an AWS-managed AWS KMS key. You can use a different encryption option by choosing **Additional configuration** in the **Data Storage** section. You can then choose the type of key to use for encryption. For more information, see [Encrypting canary artifacts \(p. 241\)](#).
14. Under **Access permissions**, choose whether to create an IAM role to run the canary or use an existing one.

If you have CloudWatch Synthetics create the role, it automatically includes all the necessary permissions. If you want to create the role yourself, see [Required roles and permissions for canaries \(p. 164\)](#) for information about the necessary permissions.

If you use the CloudWatch console to create a role for a canary when you create the canary, you can't re-use the role for other canaries, because these roles are specific to just one canary. If you have manually created a role that works for multiple canaries, you can use that existing role.

To use an existing role, you must have the `iam:PassRole` permission to pass that role to Synthetics and Lambda. You must also have the `iam:GetRole` permission.

15. (Optional) Under **Alarms**, choose whether you want default CloudWatch alarms to be created for this canary. If you choose to create alarms, they are created with the following name convention: `Synthetics-Alarm-canaryName-index`
`index` is a number representing each different alarm created for this canary. The first alarm has an index of 1, the second alarm has an index of 2, and so on.
 16. (Optional) To have this canary test an endpoint that is on a VPC, choose **VPC settings**, and then do the following:
 - a. Select the VPC that hosts the endpoint.
 - b. Select one or more subnets on your VPC. You must select a private subnet because a Lambda instance can't be configured to run in a public subnet when an IP address can't be assigned to the Lambda instance during execution. For more information, see [Configuring a Lambda Function to Access Resources in a VPC](#).
 - c. Select one or more security groups on your VPC.
- If the endpoint is on a VPC, you must enable your canary to send information to CloudWatch and Amazon S3. For more information, see [Running a canary on a VPC \(p. 240\)](#).
17. (Optional) Under **Tags**, add one or more key-value pairs as tags for this canary. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see [Tagging your Amazon CloudWatch resources \(p. 845\)](#).
 18. (Optional) Under **Active tracing**, choose whether to enable active X-Ray tracing for this canary. This option is available only if the canary uses runtime version `syn-nodejs-2.0` or later. For more information, see [Canaries and X-Ray tracing \(p. 239\)](#).

Resources that are created for canaries

When you create a canary, the following resources are created:

- An IAM role with the name `CloudWatchSyntheticsRole-canary-name-uuid` (if you use CloudWatch console to create the canary and specify for a new role to be created for the canary)
- An IAM policy with the name `CloudWatchSyntheticsPolicy-canary-name-uuid`.
- An S3 bucket with the name `cw-syn-results-accountID-region`.
- Alarms with the name `Synthetics-Alarm-MyCanaryName`, if you want alarms to be created for the canary.
- Lambda functions and layers, if you use a blueprint to create the canary. These resources have the prefix `cwsyn-MyCanaryName`.
- CloudWatch Logs log groups with the name `/aws/lambda/cwsyn-MyCanaryName`.

Using canary blueprints

This section provides details about each of the canary blueprints and the tasks each blueprint is best suited for. Blueprints are provided for the following canary types:

- Heartbeat Monitor
- API Canary
- Broken Link Checker
- Visual Monitoring
- Canary Recorder
- GUI Workflow

When you use a blueprint to create a canary, as you fill out the fields in the CloudWatch console, the **Script editor** area of the page displays the canary you are creating as a Node.js script. You can also edit your canary in this area to customize it further.

Heartbeat monitoring

Heartbeat scripts load the specified URL and store a screenshot of the page and an HTTP archive file (HAR file). They also store logs of accessed URLs.

You can use the HAR files to view detailed performance data about the web pages. You can analyze the list of web requests and catch performance issues such as time to load for an item.

If your canary uses the `syn-nodejs-puppeteer-3.1` or later runtime version, you can use the heartbeat monitoring blueprint to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

API canary

API canaries can test the basic Read and Write functions of a REST API. REST stands for *representational state transfer* and is a set of rules that developers follow when creating an API. One of these rules states that a link to a specific URL should return a piece of data.

Canaries can work with any APIs and test all types of functionality. Each canary can make multiple API calls.

In canaries that use runtime version `syn-nodejs-2.2` or later, the API canary blueprint supports multi-step canaries that monitor your APIs as HTTP steps. You can test multiple APIs in a single canary. Each step is a separate request that can access a different URL, use different headers, and use different rules for whether headers and response bodies are captured. By not capturing headers and response body, you can prevent sensitive data from being recorded.

Each request in an API canary consists of the following information:

- The *endpoint*, which is the URL that you request.
- The *method*, which is the type of request that is sent to the server. REST APIs support GET (read), POST (write), PUT (update), PATCH (update), and DELETE (delete) operations.
- The *headers*, which provide information to both the client and the server. They are used for authentication and providing information about the body content. For a list of valid headers, see [HTTP Headers](#).
- The *data (or body)*, which contains information to be sent to the server. This is used only for POST, PUT, PATCH, or DELETE requests.
- The URL that you request.

The API canary blueprint supports GET and POST methods. When you use this blueprint, you must specify headers. For example, you can specify **Authorization** as a **Key** and specify the necessary authorization data as the **Value** for that key.

If you are testing a POST request, you also specify the content to post in the **Data** field.

Integration with API Gateway

The API blueprint is integrated with Amazon API Gateway. This enables you to select an API Gateway API and stage from the same AWS account and Region as the canary, or to upload a Swagger template from API Gateway for cross-account and cross-Region API monitoring. You can then choose the rest of the details in the console to create the canary, instead of entering them from scratch. For more information about API Gateway, see [What is Amazon API Gateway?](#)

Using a private API

You can create a canary that uses a private API in Amazon API Gateway. For more information, see [Creating a private API in Amazon API Gateway?](#)

Broken link checker

The broken link checker collects all the links inside the URL that you are testing by using `document.getElementsByTagName('a')`. It tests only up to the number of links that you specify, and the URL itself is counted as the first link. For example, if you want to check all the links on a page that contains five links, you must specify for the canary to follow six links.

Broken link checker canaries created using the `syn-nodejs-2.0-beta` runtime or later support the following additional features:

- Provides a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- When viewing canary results, you can filter to see only the broken links and then fix the link based on the reason for failure.
- This version captures annotated source page screenshots for each link and highlights the anchor where the link was found. Hidden components are not annotated.
- You can configure this version to capture screenshots of both source and destination pages, just source pages, or just destination pages.
- This version fixes an issue in the earlier version where the canary script stops after the first broken link even when more links are scraped from the first page.

If you want to update an existing canary using `syn-1.0` to use the new runtime, you must delete and re-create the canary. Updating an existing canary to the new runtime does not make these features available.

A broken link checker canary detects the following types of link errors:

- 404 Page Not Found
- Invalid Host Name
- Bad URL. For example, the URL is missing a bracket, has extra slashes, or uses the wrong protocol.
- Invalid HTTP response code.
- The host server returns empty responses with no content and no response code.
- The HTTP requests constantly time out during the canary's run.
- The host consistently drops connections because it is misconfigured or is too busy.

Visual monitoring blueprint

The visual monitoring blueprint includes code to compare screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails. Visual monitoring is supported in canaries running `syn-puppeteer-node-3.2` and later. It is not currently supported in canaries running Python and Selenium.

The visual monitoring blueprint includes the following line of code in the default blueprint canary script, which enables visual monitoring.

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see [Editing or deleting a canary \(p. 246\)](#).

You can also change the canary run that is used as the baseline by using the `nextrun` or `lastrun` parameters or specifying a canary run ID in the [UpdateCanary API](#).

When you use the visual monitoring blueprint, you enter the URL where you want the screenshot to be taken, and specify a difference threshold as a percentage. After the baseline run, future runs of the canary that detect a visual difference greater than that threshold trigger a canary failure. After the baseline run, you can also edit the canary to "draw" boundaries on the baseline screenshot that you want to ignore during the visual monitoring.

The visual monitoring feature is powered by the the ImageMagick open source software toolkit. For more information, see [ImageMagick](#) .

Canary recorder

With the canary recorder blueprint, you can use the CloudWatch Synthetics Recorder to record your click and type actions on a website and automatically generate a Node.js script that can be used to create a canary that follows the same steps. The CloudWatch Synthetics Recorder is a Google Chrome extension provided by Amazon.

Credits: The CloudWatch Synthetics Recorder is based on the [Headless recorder](#) .

For more information, see [Using the CloudWatch Synthetics Recorder for Google Chrome \(p. 179\)](#).

GUI workflow builder

The GUI Workflow Builder blueprint verifies that actions can be taken on your webpage. For example, if you have a webpage with a login form, the canary can populate the user and password fields and submit the form to verify that the webpage is working correctly.

When you use a blueprint to create this type of canary, you specify the actions that you want the canary to take on the webpage. The actions that you can use are the following:

- **Click**— Selects the element that you specify and simulates a user clicking or choosing the element.

To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Verify selector**— Verifies that the specified element exists on the webpage. This test is useful for verifying that a previous action has caused the correct elements to populate the page.

To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[class=]`.

- **Verify text**— Verifies that the specified string is contained within the target element. This test is useful for verifying that a previous action has caused the correct text to be displayed.

To specify the element in a Node.js script, use a format such as `div[@id=]//h1` because this action uses the `waitForXPath` function in Puppeteer.

To specify the element in a Python script, use xpath format such as `//*[@id=]` or `//*[@class=]` because this action uses the `implicitly_wait` function in Selenium.

- **Input text**— Writes the specified text in the target element.

To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Click with navigation**— Waits for the whole page to load after choosing the specified element. This is most useful when you need to reload the page.

To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

For example, the following blueprint uses Node.js. It clicks the **firstButton** on the specified URL, verifies that the expected selector with the expected text appears, inputs the name **Test_Customer** into the **Name** field, clicks the **Login** button, and then verifies that the login is successful by checking for the **Welcome** text on the next page.

Application or endpoint URL [Info](#)

https:// ▾ www.example.com

Enter the endpoint, API or url that you are testing.

Workflow builder

Select the actions you would like the canary to take.

| Action | Selector | Text | Remove action |
|-----------------------|------------------------|---------------|---------------|
| Click | [id='firstButton'] | | Remove action |
| Verify selector | div[id='screen2Text'] | | Remove action |
| Verify text | [@id='screen2Text']/h3 | Type | Remove action |
| Input text | input[id='Name'] | Test_Customer | Remove action |
| Click with navigation | [id='Login'] | | Remove action |
| Verify text | div[@id='welcome']/h1 | Welcome | Remove action |

Add action

GUI workflow canaries that use the following runtimes also provide a summary of the steps executed for each canary run. You can use the screenshots and error message associated with each step to find the root cause of failure.

- `syn-nodejs-2.0` or later
- `syn-python-selenium-1.0` or later

Using the CloudWatch Synthetics Recorder for Google Chrome

Amazon provides a CloudWatch Synthetics Recorder to help you create canaries more easily. The recorder is a Google Chrome extension.

The recorder records your click and type actions on a website and automatically generates a Node.js script that can be used to create a canary that follows the same steps.

After you start recording, the CloudWatch Synthetics Recorder detects your actions in the browser and converts them to a script. You can pause and resume the recording as needed. When you stop recording, the recorder produces a Node.js script of your actions, which you can easily copy with the **Copy to Clipboard** button. You can then use this script to create a canary in CloudWatch Synthetics.

Credits: The CloudWatch Synthetics Recorder is based on the [Headless recorder](#).

Installing the CloudWatch Synthetics Recorder extension for Google Chrome

To use the CloudWatch Synthetics Recorder, you can start creating a canary and choose the **Canary Recorder** blueprint. If you do this when you haven't already downloaded the recorder, the CloudWatch Synthetics console provides a link to download it.

Alternatively, you can follow these steps to download and install the recorder directly.

To install the CloudWatch Synthetics Recorder

1. Using Google Chrome, go to this website: <https://chrome.google.com/webstore/detail/cloudwatch-synthetics-rec/bhdnlmmgiplmbcdmkdfplenecegfno>
2. Choose **Add to Chrome**, then choose **Add extension**.

Using the CloudWatch Synthetics Recorder for Google Chrome

To use the CloudWatch Synthetics Recorder to help you create a canary, you can choose **Create canary** in the CloudWatch console, and then choose **Use a blueprint**, **Canary Recorder**. For more information, see [Creating a canary \(p. 173\)](#).

Alternatively, you can use the recorder to record steps without immediately using them to create a canary.

To use the CloudWatch Synthetics Recorder to record your actions on a website

1. Navigate to the page that you want to monitor.
2. Choose the Chrome extensions icon, and then choose **CloudWatch Synthetics Recorder**.
3. Choose **Start Recording**.
4. Perform the steps that you want to record. To pause recording, choose **Pause**.
5. When you are finished recording the workflow, choose **Stop recording**.
6. Choose **Copy to clipboard** to copy the generated script to your clipboard. Or, if you want to start over, choose **New recording**.
7. To create a canary with the copied script, you can paste your copied script into the recorder blueprint inline editor, or save it to an Amazon S3 bucket and import it from there.
8. If you're not immediately creating a canary, you can save your recorded script to a file.

Known limitations of the CloudWatch Synthetics Recorder

The CloudWatch Synthetics Recorder for Google Chrome currently has the following limitations.

- HTML elements that don't have IDs will use CSS selectors. This can break canaries if the webpage structure changes later. We plan to provide some configuration options (such as using data-id) around this in a future version of the recorder.

- The recorder doesn't support actions such as double-click or copy/paste, and doesn't support key combinations such as CMD+O.
- To verify the presence of an element or text on the page, users must add assertions after the script is generated. The recorder doesn't support verifying an element without performing any action on that element. This is similar to the "Verify text" or "Verify element" options in the canary workflow builder. We plan to add some assertions support in a future version of the recorder.
- The recorder records all actions in the tab where the recording is initiated. It doesn't record pop-ups (for instance, to allow location tracking) or navigation to different pages from pop-ups.

Synthetics runtime versions

When you create or update a canary, you choose a Synthetics runtime version for the canary. A Synthetics runtime is a combination of the Synthetics code that calls your script handler, and the Lambda layers of bundled dependencies.

CloudWatch Synthetics currently supports runtimes that use Node.js for scripts and the Puppeteer framework, and runtimes that use Python for scripting and Selenium Webdriver for the framework.

We recommend that you always use the most recent runtime version for your canaries, to be able to use the latest features and updates made to the Synthetics library.

Topics

- [CloudWatch Synthetics runtime support policy \(p. 181\)](#)
- [Runtime versions using Node.js and Puppeteer \(p. 185\)](#)
- [Runtime versions using Python and Selenium Webdriver \(p. 191\)](#)

CloudWatch Synthetics runtime support policy

Synthetics runtime versions are subject to maintenance and security updates. When any component of a runtime version is no longer supported, that Synthetics runtime version is deprecated.

You can't create canaries using deprecated runtime versions. Canaries that use deprecated runtimes continue to run. You can stop, start, and delete these canaries. You can update an existing canary that uses a deprecated runtime version by updating the canary to use a supported runtime version.

CloudWatch Synthetics notifies you by email if you have canaries that use runtimes that are scheduled to be deprecated in the next 60 days. We recommend that you migrate your canaries to a supported runtime version to benefit from the new functionality, security, and performance enhancements that are included in more recent releases.

How do I update a canary to a new runtime version?

You can update a canary's runtime version by using the CloudWatch console, AWS CloudFormation, the AWS CLI or the AWS SDK. When you use the CloudWatch console, you can update up to five canaries at once by selecting them in the canary list page and then choosing **Actions, Update Runtime**.

You can verify the upgrade by first cloning the canary using the CloudWatch console and updating its runtime version. This creates another canary which is a clone of your original canary. Once you have verified your canary with the new runtime version, you can update the runtime version of your original canary and delete the clone canary.

You can also update multiple canaries using an upgrade script. For more information, see [Canary runtime upgrade script \(p. 182\)](#).

If you upgrade a canary and it fails, see [Troubleshooting a failed canary \(p. 232\)](#).

Runtime deprecation dates

| Runtime Version | Deprecation date |
|---------------------|------------------|
| syn-nodejs-2.2 | May 28, 2021 |
| syn-nodejs-2.1 | May 28, 2021 |
| syn-nodejs-2.0 | May 28, 2021 |
| syn-nodejs-2.0-beta | February 8, 2021 |
| syn-1.0 | May 28, 2021 |

Canary runtime upgrade script

To upgrade a canary script to a supported runtime version, use the following script.

```
const AWS = require('aws-sdk');

// You need to configure your AWS credentials and Region.
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-credentials-node.html
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-region.html

const synthetics = new AWS.Synthetics();

const DEFAULT_OPTIONS = {
    /**
     * The number of canaries to upgrade during a single run of this script.
     */
    count: 10,
    /**
     * No canaries are upgraded unless force is specified.
     */
    force: false
};

/**
 * The number of milliseconds to sleep between GetCanary calls when
 * verifying that an update succeeded.
 */
const SLEEP_TIME = 5000;

(async () => {
    try {
        const options = getOptions();

        const versions = await getRuntimeVersions();
        const canaries = await getAllCanaries();
        const upgrades = canaries
            .filter(canary => !versions.isLatestVersion(canary.RuntimeVersion))
            .map(canary => {
                return {
                    Name: canary.Name,
                    FromVersion: canary.RuntimeVersion,
                    ToVersion: versions.getLatestVersion(canary.RuntimeVersion)
                };
            });
    }

    if (options.force) {
```

```

const promises = [];

for (const upgrade of upgrades.slice(0, options.count)) {
    const promise = upgradeCanary(upgrade);
    promises.push(promise);
    // Sleep for 100 milliseconds to avoid throttling.
    await usleep(100);
}

const succeeded = [];
const failed = [];
for (let i = 0; i < upgrades.slice(0, options.count).length; i++) {
    const upgrade = upgrades[i];
    const promise = promises[i];
    try {
        await promise;
        console.log(`The update of ${upgrade.Name} succeeded.`);
        succeeded.push(upgrade.Name);
    } catch (e) {
        console.log(`The update of ${upgrade.Name} failed with error: ${e}`);
        failed.push({
            Name: upgrade.Name,
            Reason: e
        });
    }
}

if (succeeded.length) {
    console.group('The following canaries were upgraded successfully.');
    for (const name of succeeded) {
        console.log(name);
    }
    console.groupEnd()
} else {
    console.log('No canaries were upgraded successfully.');
}

if (failed.length) {
    console.group('The following canaries were not upgraded successfully.');
    for (const failure of failed) {
        console.log(`\x1b[31m${failure.Name}: ${failure.Reason}\x1b[0m`);
    }
    console.groupEnd();
}
} else {
    console.log('Run with --force [--count <count>] to perform the first <count> upgrades shown. The default value of <count> is 10.')
    console.table(upgrades);
}
} catch (e) {
    console.error(e);
}
})();

function getOptions() {
    const force = getFlag('--force', DEFAULT_OPTIONS.force);
    const count = getOption('--count', DEFAULT_OPTIONS.count);
    return { force, count };
}

function getFlag(key, defaultValue) {
    return process.argv.includes(key) || defaultValue;
}
function getOption(key, defaultValue) {
    const index = process.argv.indexOf(key);
    if (index < 0) {
        return defaultValue;
    }
}

```

```

        }
        const value = process.argv[index + 1];
        if (typeof value === 'undefined' || value.startsWith('-')) {
            throw `The ${key} option requires a value.`;
        }
        return value;
    }
}

function getAllCanaries() {
    return new Promise((resolve, reject) => {
        const canaries = [];

        synthetics.describeCanaries().eachPage((err, data) => {
            if (err) {
                reject(err);
            } else {
                if (data === null) {
                    resolve(canaries);
                } else {
                    canaries.push(...data.Canaries);
                }
            }
        });
    });
}

function getRuntimeVersions() {
    return new Promise((resolve, reject) => {
        const jsVersions = [];
        const pythonVersions = [];
        synthetics.describeRuntimeVersions().eachPage((err, data) => {
            if (err) {
                reject(err);
            } else {
                if (data === null) {
                    jsVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
                    pythonVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
                    resolve({
                        isLatestVersion(version) {
                            const latest = this.getLatestVersion(version);
                            return latest === version;
                        },
                        getLatestVersion(version) {
                            if (jsVersions.some(v => v.VersionName === version)) {
                                return jsVersions[jsVersions.length - 1].VersionName;
                            } else if (pythonVersions.some(v => v.VersionName === version)) {
                                return pythonVersions[pythonVersions.length - 1].VersionName;
                            } else {
                                throw Error(`Unknown version ${version}`);
                            }
                        }
                    });
                } else {
                    for (const version of data.RuntimeVersions) {
                        if (version.VersionName === 'syn-1.0') {
                            jsVersions.push(version);
                        } else if (version.VersionName.startsWith('syn-nodejs-2.')) {
                            jsVersions.push(version);
                        } else if (version.VersionName.startsWith('syn-nodejs-puppeteer-')) {
                            jsVersions.push(version);
                        } else if (version.VersionName.startsWith('syn-python-selenium-')) {
                            pythonVersions.push(version);
                        } else {
                            throw Error(`Unknown version ${version.VersionName}`);
                        }
                    }
                }
            }
        });
    });
}

```

```

        }
    }
});

}

async function upgradeCanary(upgrade) {
    console.log(`Upgrading canary ${upgrade.Name} from ${upgrade.FromVersion} to
${upgrade.ToVersion}`);
    await synthetics.updateCanary({ Name: upgrade.Name, RuntimeVersion:
upgrade.ToVersion }).promise();
    while (true) {
        await usleep(SLEEP_TIME);
        console.log(`Getting the state of canary ${upgrade.Name}`);
        const response = await synthetics.getCanary({ Name: upgrade.Name }).promise();
        const state = response.Canary.Status.State;
        console.log(`The state of canary ${upgrade.Name} is ${state}`);
        if (state === 'ERROR' || response.Canary.Status.StateReason) {
            throw response.Canary.Status.StateReason;
        }
        if (state !== 'UPDATING') {
            return;
        }
    }
}

function usleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}

```

Runtime versions using Node.js and Puppeteer

The first runtime version for Node.js and Puppeteer was named syn-1.0. Later runtime

versions have the naming convention syn-*language-majorversion.minorversion*.

Starting with syn-nodejs-puppeteer-3.0, the naming convention is
syn-*language-framework-majorversion.minorversion*

An additional -beta suffix shows that the runtime version is currently in a beta preview release.

Runtime versions with the same major version number are backward compatible.

Notes for all runtime versions

When using syn-nodejs-puppeteer-3.0 runtime version, make sure that your canary script is compatible with Node.js 12.x. If you use an earlier version of a syn-nodejs runtime version, make sure that that your script is compatible with Node.js 10.x.

The Lambda code in a canary is configured to have a maximum memory of 1 GB. Each run of a canary times out after a configured timeout value. If no timeout value is specified for a canary, CloudWatch chooses a timeout value based on the canary's frequency. If you configure a timeout value, make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

[syn-nodejs-puppeteer-3.5](#)

The syn-nodejs-puppeteer-3.5 runtime is the newest runtime version for Node.js and Puppeteer.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 10.1.0

- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-3.5:

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

[syn-nodejs-puppeteer-3.4](#)

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.4:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.

- **Expanded HAR file information**— You can now see bad, pending, and incomplete requests in the HAR files produced by canaries.

[syn-nodejs-puppeteer-3.3](#)

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.3:

- **More options for artifact encryption**— For canaries using this runtime or later, instead of using an AWS managed key to encrypt artifacts that the canary stores in Amazon S3, you can choose to use an AWS KMS customer managed key or an Amazon S3-managed key. For more information, see [Encrypting canary artifacts \(p. 241\)](#).

[syn-nodejs-puppeteer-3.2](#)

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.2:

- **visual monitoring with screenshots**— Canaries using this runtime or later can compare a screenshot taken during a run with a baseline version of the same screenshot. If the screenshots are more

different than a specified percentage threshold, the canary fails. For more information, see [Visual monitoring \(p. 209\)](#) or [Visual monitoring blueprint \(p. 177\)](#).

- **New functions regarding sensitive data** You can prevent sensitive data from appearing in canary logs and reports. For more information, see [SyntheticsLogHelper class \(p. 210\)](#).
- **Deprecated function** The `RequestResponseLogHelper` class is deprecated in favor of other new configuration options. For more information, see [RequestResponseLogHelper class \(p. 216\)](#).

[syn-nodejs-puppeteer-3.1](#)

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.1:

- **Ability to configure CloudWatch metrics**— With this runtime, you can disable the metrics that you do not require. Otherwise, canaries publish various CloudWatch metrics for each canary run.
- **Screenshot linking**— You can link a screenshot to a canary step after the step has completed. To do this, you take the screenshot by using the `takeScreenshot` method, using the name of the step that you want to associate the screenshot with. For example, you might want to perform a step, add a wait time, and then take the screenshot.
- **Heartbeat monitor blueprint can monitor multiple URLs**— You can use the heartbeat monitoring blueprint in the CloudWatch console to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

[syn-nodejs-puppeteer-3.0](#)

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.0:

- **Upgraded dependencies**— This version uses Puppeteer version 5.5.0, Node.js 12.x, and Chromium 88.0.4298.0.
- **Cross-Region bucket access**— You can now specify an S3 bucket in another Region as the bucket where your canary stores its log files, screenshots, and HAR files.
- **New functions available**— This version adds library functions to retrieve the canary name and the Synthetics runtime version.

For more information, see [Synthetics class \(p. 200\)](#).

[syn-nodejs-2.2](#)

This section contains information about the `syn-nodejs-2.2` runtime version.

Important

This runtime version is scheduled to be deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy \(p. 181\)](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in syn-nodejs-2.2:

- **Monitor your canaries as HTTP steps**— You can now test multiple APIs in a single canary. Each API is tested as a separate HTTP step, and CloudWatch Synthetics monitors the status of each step using step metrics and the CloudWatch Synthetics step report. CloudWatch Synthetics creates SuccessPercent and Duration metrics for each HTTP step.

This functionality is implemented by the [executeHttpStep\(stepName, requestOptions, callback, stepConfig\)](#) function. For more information, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\) \(p. 220\)](#).

The API canary blueprint is updated to use this new feature.

- **HTTP request reporting**— You can now view detailed HTTP requests reports which capture details such as request/response headers, response body, status code, error and performance timings, TCP connection time, TLS handshake time, first byte time, and content transfer time. All HTTP requests which use the HTTP/HTTPS module under the hood are captured here. Headers and response body are not captured by default but can be enabled by setting configuration options.
- **Global and step-level configuration**— You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key/value pairs to enable or disable certain options.

For more information, see [SyntheticsConfiguration class \(p. 201\)](#).

- **Continue on step failure configuration**— You can choose to continue canary execution when a step fails. For the executeHttpStep function, this is turned on by default. You can set this option once at global level or set it differently per-step.

syn-nodejs-2.1

Important

This runtime version is scheduled to be deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy \(p. 181\)](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in syn-nodejs-2.1:

- **Configurable screenshot behavior**— Provides the ability to turn off the capturing of screenshots by UI canaries. In canaries that use previous versions of the runtimes, UI canaries always capture screenshots before and after each step. With syn-nodejs-2.1, this is configurable. Turning off screenshots can reduce your Amazon S3 storage costs, and can help you comply with HIPAA regulations. For more information, see [SyntheticsConfiguration class \(p. 201\)](#).
- **Customize the Google Chrome launch parameters** You can now configure the arguments used when a canary launches a Google Chrome browser window. For more information, see [launch\(options\) \(p. 215\)](#).

There can be a small increase in canary duration when using syn-nodejs-2.0 or later, compared to earlier versions of the canary runtimes.

[syn-nodejs-2.0](#)

Important

This runtime version is scheduled to be deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy \(p. 181\)](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in syn-nodejs-2.0:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Support for X-Ray active tracing.** When a canary has tracing enabled, X-Ray traces are sent for all calls made by the canary that use the browser, the AWS SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the service map in both CloudWatch ServiceLens and in X-Ray, even when they don't send requests to other services or applications that have tracing enabled. For more information, see [Canaries and X-Ray tracing \(p. 239\)](#).
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.
- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. With this runtime version, these metrics are reported only for UI canaries, and are not reported for API canaries. They are also reported for API canaries starting with runtime version `syn-nodejs-puppeteer-2.2`.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.
- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in syn-nodejs-2.0:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

There can be a small increase in canary duration when using syn-nodejs-2.0 or later, compared to earlier versions of the canary runtimes.

syn-nodejs-2.0-beta

Important

This runtime version was deprecated on February 8, 2021. For more information, see [CloudWatch Synthetics runtime support policy \(p. 181\)](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in syn-nodejs-2.0-beta:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.
- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. These metrics are reported only for UI canaries, and are not reported for API canaries.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.
- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in syn-nodejs-2.0-beta:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

syn-1.0

Important

This runtime version is scheduled to be deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy \(p. 181\)](#).

The first Synthetics runtime version is `syn-1.0`.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 1.14.0
- The Chromium version that matches Puppeteer-core 1.14.0

Runtime versions using Python and Selenium Webdriver

The following sections contain information about the CloudWatch Synthetics runtime versions for Python and Selenium Webdriver. Selenium is an open-source browser automation tool. For more information about Selenium, see www.selenium.dev/

The naming convention for these runtime versions is `syn-language-framework-majorversion.minorversion`.

[syn-python-selenium-1.2](#)

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

[syn-python-selenium-1.1](#)

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.

- **Configuration options for adding metrics and step failure configurations**— These options were already available in runtimes for Node.js canaries. For more information, see [SyntheticsConfiguration class \(p. 222\)](#).
- **Custom arguments in Chrome**— You can now open a browser in incognito mode or pass in proxy server configuration. For more information, see [Chrome\(\) \(p. 230\)](#).
- **Cross-Region artifact buckets**— A canary can store its artifacts in an Amazon S3 bucket in a different Region.
- **Bug fixes, including a fix for the `index.py` issue**— With previous runtimes, a canary file named `index.py` caused exceptions because it conflicted with the name of the library file. This issue is now fixed.

[syn-python-selenium-1.0](#)

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Selenium support**— You can write canary scripts using the Selenium test framework. You can bring your Selenium scripts from elsewhere into CloudWatch Synthetics with minimal changes, and they will work with AWS services.

Writing a canary script

The following sections explain how to write a canary script and how to integrate a canary with other AWS Services.

Topics

- [Writing a Node.js canary script \(p. 192\)](#)
- [Writing a Python canary script \(p. 198\)](#)
- [Changing an existing Selenium script to use a Synthetics canary \(p. 199\)](#)

Writing a Node.js canary script

Topics

- [Creating a CloudWatch Synthetics canary from scratch \(p. 192\)](#)
- [Packaging your canary files \(p. 193\)](#)
- [Changing an existing Puppeteer script to use as a Synthetics canary \(p. 194\)](#)
- [Environment variables \(p. 195\)](#)
- [Integrating your canary with other AWS services \(p. 196\)](#)
- [Forcing your canary to use a static IP address \(p. 198\)](#)

Creating a CloudWatch Synthetics canary from scratch

Here is an example minimal Synthetics Canary script. This script passes as a successful run, and returns a string. To see what a failing canary looks like, change `let fail = false;` to `let fail = true;`.

You must define an entry point function for the canary script. To see how files are uploaded to the Amazon S3 location specified as the canary's `ArtifactS3Location`, create these files under the `/tmp` folder. After the script runs, the pass/fail status and the duration metrics are published to CloudWatch and the files under `/tmp` are uploaded to S3.

```
const basicCustomEntryPoint = async function () {  
  
    // Insert your code here  
  
    // Perform multi-step pass/fail check  
  
    // Log decisions made and results to /tmp  
  
    // Be sure to wait for all your code paths to complete  
    // before returning control back to Synthetics.  
    // In that way, your canary will not finish and report success  
    // before your code has finished executing  
  
    // Throw to fail, return to succeed  
    let fail = false;  
    if (fail) {  
        throw "Failed basicCanary check.";
```

```

        }

        return "Successfully completed basicCanary checks.";
    };

exports.handler = async () => {
    return await basicCustomEntryPoint();
};

```

Next, we'll expand the script to use Synthetics logging and make a call using the AWS SDK. For demonstration purposes, this script will create an Amazon DynamoDB client and make a call to the DynamoDB listTables API. It logs the response to the request and logs either pass or fail depending on whether the request was successful.

```

const log = require('SyntheticsLogger');
const AWS = require('aws-sdk');
// Require any dependencies that your script needs
// Bundle additional files and dependencies into a .zip file with folder structure
// nodejs/node_modules/additional files and folders

const basicCustomEntryPoint = async function () {

    log.info("Starting DynamoDB:listTables canary.");

    let dynamodb = new AWS.DynamoDB();
    var params = {};
    let request = await dynamodb.listTables(params);
    try {
        let response = await request.promise();
        log.info("listTables response: " + JSON.stringify(response));
    } catch (err) {
        log.error("listTables error: " + JSON.stringify(err), err.stack);
        throw err;
    }

    return "Successfully completed DynamoDB:listTables canary.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};

```

Packaging your canary files

If you are uploading your canary scripts using an Amazon S3 location, your zip file should include your script under this folder structure: nodejs/node_modules/**myCanaryFilename.js file**.

If you have more than a single .js file or you have a dependency that your script depends on, you can bundle them all into a single ZIP file that contains the folder structure nodejs/node_modules/**myCanaryFilename.js file and other folders and files**. If you are using syn-nodejs-puppeteer-3.4 or later, you can optionally put your canary files in another folder and creating your folder structure like this: nodejs/node_modules/**myFolder/myCanaryFilename.js file and other folders and files**.

Handler name

Be sure to set your canary's script entry point (handler) as `myCanaryFilename.functionName` to match the file name of your script's entry point. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, then `functionName` must be `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, you can choose any function name as the handler. If you are using `syn-nodejs-puppeteer-3.4` or later, you can also optionally store the canary in a separate folder such as

`nodejs/node_modules/myFolder/my_canary_filename`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Puppeteer script to use as a Synthetics canary

This section explains how to take Puppeteer scripts and modify them to run as Synthetics canary scripts. For more information about Puppeteer, see [Puppeteer API v1.14.0](#).

We'll start with this example Puppeteer script:

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

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

The conversion steps are as follows:

- Create and export a handler function. The handler is the entry point function for the script. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, the handler function must be named `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-nodejs-puppeteer-3.4` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

```
const basicPuppeteerExample = async function () {};

exports.handler = async () => {
  return await basicPuppeteerExample();
};
```

- Use the `Synthetics` dependency.

```
var synthetics = require('Synthetics');
```

- Use the `Synthetics.getPage` function to get a Puppeteer Page object.

```
const page = await synthetics.getPage();
```

The page object returned by the `Synthetics.getPage` function has the `page.on` request, response and `requestfailed` events instrumented for logging. Synthetics also sets up HAR file generation for requests and responses on the page, and adds the canary ARN to the user-agent headers of outgoing requests on the page.

The script is now ready to be run as a Synthetics canary. Here is the updated script:

```
var synthetics = require('Synthetics'); // Synthetics dependency

const basicPuppeteerExample = async function () {
  const page = await synthetics.getPage(); // Get instrumented page from Synthetics
  await page.goto('https://example.com');
  await page.screenshot({path: '/tmp/example.png'}); // Write screenshot to /tmp folder
};
```

```
exports.handler = async () => { // Exported handler function
    return await basicPuppeteerExample();
};
```

Environment variables

You can use environment variables when creating canaries. This allows you to write a single canary script and then use that script with different values to quickly create multiple canaries that have a similar task.

For example, suppose your organization has endpoints such as prod, dev, and pre-release for the different stages of your software development, and you need to create canaries to test each of these endpoints. You can write a single canary script that tests your software and then specify different values for the endpoint environment variable when you create each of the three canaries. Then, when you create a canary, you specify the script and the values to use for the environment variables.

The names of environment variables can contain letters, numbers, and the underscore character. They must start with a letter and be at least two characters. The total size of your environment variables can't exceed 4 KB. You can't specify any Lambda reserved environment variables as the names of your environment variables. For more information about reserved environment variables, see [Runtime environment variables](#).

The following example script uses two environment variables. This script is for a canary that checks whether a webpage is available. It uses environment variables to parameterize both the URL that it checks and the CloudWatch Synthetics log level that it uses.

The following function sets `LogLevel` to the value of the `LOG_LEVEL` environment variable.

```
synthetics.setLevel(process.env.LOG_LEVEL);
```

This function sets `URL` to the value of the `URL` environment variable.

```
const URL = process.env.URL;
```

This is the complete script. When you create a canary using this script, you specify values for the `LOG_LEVEL` and `URL` environment variables.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadEnvironmentVariable = async function () {

    // Setting the log level (0-3)
    synthetics.setLevel(process.env.LOG_LEVEL);
    // INSERT URL here
    const URL = process.env.URL;

    let page = await synthetics.getPage();
    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout: 30000});
    if (!response) {
        throw "Failed to load page!";
    }
    //Wait for page to render.
    //Increase or decrease wait time based on endpoint being monitored.
    await page.waitFor(15000);
    await synthetics.takeScreenshot('loaded', 'loaded');
    let pageTitle = await page.title();
```

```
    log.info('Page title: ' + pageTitle);
    log.debug('Environment variable:' + process.env.URL);

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadEnvironmentVariable();
};
```

Passing environment variables to your script

To pass environment variables to your script when you create a canary in the console, specify the keys and values of the environment variables in the **Environment variables** section on the console. For more information, see [Creating a canary \(p. 173\)](#).

To pass environment variables through the API or AWS CLI, use the `EnvironmentVariables` parameter in the `RunConfig` section. The following is an example AWS CLI command that creates a canary that uses two environment variables with keys of `Environment` and `Region`.

```
aws synthetics create-canary --cli-input-json '{
    "Name": "nameofCanary",
    "ExecutionRoleArn": "roleArn",
    "ArtifactS3Location": "s3://cw-syn-results-123456789012-us-west-2",
    "Schedule": {
        "Expression": "rate(0 minute)",
        "DurationInSeconds": 604800
    },
    "Code": {
        "S3Bucket": "canarycreation",
        "S3Key": "cwsyn-mycanaryheartbeat-12345678-d1bd-1234-
abcd-123456789012-12345678-6a1f-47c3-b291-123456789012.zip",
        "Handler": "pageLoadBlueprint.handler"
    },
    "RunConfig": {
        "TimeoutInSeconds": 60,
        "EnvironmentVariables": {
            "Environment": "Production",
            "Region": "us-west-1"
        }
    },
    "SuccessRetentionPeriodInDays": 13,
    "FailureRetentionPeriodInDays": 13,
    "RuntimeVersion": "syn-nodejs-2.0"
}'
```

Integrating your canary with other AWS services

All canaries can use the AWS SDK library. You can use this library when you write your canary to integrate the canary with other AWS services.

To do so, you need to add the following code to your canary. AWS For these examples, AWS Secrets Manager is used as the service that the canary is integrating with.

- Import the AWS SDK.

```
const AWS = require('aws-sdk');
```

- Create a client for the AWS service that you are integrating with.

```
const secretsManager = new AWS.SecretsManager();
```

- Use the client to make API calls to that service.

```
var params = {
  SecretId: secretName
};
return await secretsManager.getSecretValue(params).promise();
```

The following canary script code snippet demonstrates an example of integration with Secrets Manager in more detail.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const AWS = require('aws-sdk');
const secretsManager = new AWS.SecretsManager();

const getSecrets = async (secretName) => {
  var params = {
    SecretId: secretName
  };
  return await secretsManager.getSecretValue(params).promise();
}

const secretsExample = async function () {
  let URL = "<URL>";
  let page = await synthetics.getPage();

  log.info(`Navigating to URL: ${URL}`);
  const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout: 30000});

  // Fetch secrets
  let secrets = await getSecrets("secretname")

  /**
   * Use secrets to login.
   *
   * Assuming secrets are stored in a JSON format like:
   * {
   *   "username": "<USERNAME>",
   *   "password": "<PASSWORD>"
   * }
   */
  let secretsObj = JSON.parse(secrets.SecretString);
  await synthetics.executeStep('login', async function () {
    await page.type(">USERNAME-INPUT-SELECTOR<", secretsObj.username);
    await page.type(">PASSWORD-INPUT-SELECTOR<", secretsObj.password);

    await Promise.all([
      page.waitForNavigation({ timeout: 30000 }),
      await page.click(">SUBMIT-BUTTON-SELECTOR<")
    ]);
  });

  // Verify login was successful
  await synthetics.executeStep('verify', async function () {
    await page.waitForXPath(">SELECTOR<", { timeout: 30000 });
  });
};
```

```
exports.handler = async () => {
    return await secretsExample();
};
```

Forcing your canary to use a static IP address

You can set up a canary so that it uses a static IP address.

To force a canary to use a static IP address

1. Create a new VPC. For more information, see [Using DNS with Your VPC](#).
2. Create a new internet gateway. For more information, see [Adding an internet gateway to your VPC](#).
3. Create a public subnet inside your new VPC.
4. Add a new route table to the VPC.
5. Add a route in the new route table, that goes from 0.0.0.0/0 to the internet gateway.
6. Associate the new route table with the public subnet.
7. Create an elastic IP address. For more information, see [Elastic IP addresses](#).
8. Create a new NAT gateway and assign it to the public subnet and the elastic IP address.
9. Create a private subnet inside the VPC.
10. Add a route to the VPC default route table, that goes from 0.0.0.0/0 to the NAT gateway
11. Create your canary.

Writing a Python canary script

This script passes as a successful run, and returns a string. To see what a failing canary looks like, change fail = False to fail = True

```
def basic_custom_script():
    # Insert your code here
    # Perform multi-step pass/fail check
    # Log decisions made and results to /tmp
    # Be sure to wait for all your code paths to complete
    # before returning control back to Synthetics.
    # In that way, your canary will not finish and report success
    # before your code has finished executing
    fail = False
    if fail:
        raise Exception("Failed basicCanary check.")
    return "Successfully completed basicCanary checks."
def handler(event, context):
    return basic_custom_script()
```

Packaging your canary files

If you have more than one .py file or your script has a dependency, you can bundle them all into a single ZIP file. If you use the syn-python-selenium-1.1 runtime, the ZIP file must contain your main canary .py file within a python folder, such as `python/my_canary_filename.py`. If you use syn-python-selenium-1.1 or later, you can optionally use a different folder, such as `python/myFolder/my_canary_filename.py`.

This ZIP file should contain all necessary folders and files, but the other files do not need to be in the `python` folder.

Be sure to set your canary's script entry point as `my_canary_filename.functionName` to match the file name and function name of your script's entry point. If you are using the `syn-nodejs-selenium-1.0` runtime, then `functionName` must be `handler`. If you are using `syn-nodejs-selenium-1.1` or later, this handler name restriction doesn't apply, and you can also optionally store the canary in a separate folder such as `python/myFolder/my_canary_filename.py`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Selenium script to use a Synthetics canary

You can quickly modify an existing script for Python and Selenium to be used as a canary. For more information about Selenium, see www.selenium.dev/.

For this example, we'll start with the following Selenium script:

```
from selenium import webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

basic_selenium_script()
```

The conversion steps are as follows.

To convert a Selenium script to be used as a canary

1. Change the `import` statement to use Selenium from the `aws_synthetics` module:

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
```

The Selenium module from `aws_synthetics` ensures that the canary can emit metrics and logs, generate a HAR file, and work with other CloudWatch Synthetics features.

2. Create a handler function and call your Selenium method. The handler is the entry point function for the script.

If you are using `syn-python-selenium-1.0`, the handler function must be named `handler`. If you are using `syn-python-selenium-1.1` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-python-selenium-1.1` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

```
def handler(event, context):
    basic_selenium_script()
```

The script is now updated to be a CloudWatch Synthetics canary. Here is the updated script:

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

def handler(event, context):
    basic_selenium_script()
```

Library functions available for canary scripts

CloudWatch Synthetics includes several built-in classes and functions that you can call when writing Node.js scripts to use as canaries.

Some apply to both UI and API canaries. Others apply to UI canaries only. A UI canary is a canary that uses the `getPage()` function and uses Puppeteer as a web driver to navigate and interact with webpages.

Topics

- [Library functions available for Node.js canary scripts \(p. 200\)](#)
- [Library functions available for Python canary scripts using Selenium \(p. 222\)](#)

Library functions available for Node.js canary scripts

This section lists the library functions available for Node.js canary scripts.

Topics

- [Node.js library classes and functions that apply to all canaries \(p. 200\)](#)
- [Node.js library classes and functions that apply to UI canaries only \(p. 213\)](#)
- [Node.js library classes and functions that apply to API canaries only \(p. 220\)](#)

Node.js library classes and functions that apply to all canaries

The following CloudWatch Synthetics library functions for Node.js are useful for all canaries.

Topics

- [Synthetics class \(p. 200\)](#)
- [SyntheticsConfiguration class \(p. 201\)](#)
- [Synthetics logger \(p. 209\)](#)
- [SyntheticsLogHelper class \(p. 210\)](#)

Synthetics class

The following functions for all canaries are in the `Synthetics` class.

`addExecutionError(errorMessage, ex);`

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `addExecutionError` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `addExecutionError` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try {
```

```
    await synthetics.takeScreenshot(stepName, "loaded");
} catch(ex) {
    synthetics.addExecutionError('Unable to take screenshot ', ex);
}
```

[getCanaryName\(\)](#)

Returns the name of the canary.

[getRuntimeVersion\(\)](#)

This function is available in runtime version `syn-nodejs-puppeteer-3.0` and later. It returns the Synthetics runtime version of the canary. For example, the return value could be `syn-nodejs-puppeteer-3.0`.

[getLogLevel\(\)](#)

Retrieves the current log level for the Synthetics library. Possible values are the following:

- 0 – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
let logLevel = synthetics.getLogLevel();
```

[setLogLevel\(\)](#)

Sets the log level for the Synthetics library. Possible values are the following:

- 0 – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
synthetics.setLogLevel(0);
```

[SyntheticsConfiguration class](#)

This class is available only in the `syn-nodejs-2.1` runtime version or later.

You can use the `SyntheticsConfiguration` class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key/value pairs.

You can pass in options at the step level. For examples, see [async executeStep\(stepName, functionToExecute, \[stepConfig\]\); \(p. 213\)](#) and [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\) \(p. 220\)](#)

Function definitions:

[setConfig\(options\)](#)

options is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *options*.

[setConfig\(options\) for all canaries](#)

For canaries using `syn-nodejs-puppeteer-3.2` or later, the **(options)** for `setConfig` can include the following parameters:

- `includeRequestHeaders` (boolean)— Whether to include request headers in the report. The default is `false`.
- `includeResponseHeaders` (boolean)— Whether to include response headers in the report. The default is `false`.
- `restrictedHeaders` (array)— A list of header values to ignore, if headers are included. This applies to both request and response headers. For example, you can hide your credentials by passing `includeRequestHeaders` as `true` and `restrictedHeaders` as `['Authorization']`.
- `includeRequestBody` (boolean)— Whether to include the request body in the report. The default is `false`.
- `includeResponseBody` (boolean)— Whether to include the response body in the report. The default is `false`.

[setConfig\(options\) regarding CloudWatch metrics](#)

For canaries using `syn-nodejs-puppeteer-3.1` or later, the **(options)** for `setConfig` can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with aggregated determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see [CloudWatch metrics published by canaries \(p. 244\)](#).

- `failedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failedRequestsMetric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_2xxMetric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_4xxMetric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_5xxMetric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `stepDurationMetric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.
- `stepSuccessMetric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.
- `aggregatedFailedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedRequestsMetric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

- `aggregated2xxMetric` (boolean)— Whether to emit the 2xx metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated4xxMetric` (boolean)— Whether to emit the 4xx metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated5xxMetric` (boolean)— Whether to emit the 5xx metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `visualMonitoringSuccessPercentMetric` (boolean)— Whether to emit the `visualMonitoringSuccessPercent` metric for this canary. The default is `true`.
- `visualMonitoringTotalComparisonsMetric` (boolean)— Whether to emit the `visualMonitoringTotalComparisons` metric for this canary. The default is `false`.
- `stepsReport` (boolean)— Whether to report a step execution summary. The default is `true`.
- `includeUrlPassword` (boolean)— Whether to include a password that appears in the URL. By default, passwords that appear in URLs are redacted from logs and reports, to prevent disclosing sensitive data. The default is `false`.
- `restrictedUrlParameters` (array)— A list of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. The parameter is case-insensitive. You can pass an asterisk (*) as a value to redact all URL path and query parameter values. The default is an empty array.
- `logRequest` (boolean)— Whether to log every request in canary logs. For UI canaries, this logs each request sent by the browser. The default is `true`.
- `logResponse` (boolean)— Whether to log every response in canary logs. For UI canaries, this logs every response received by the browser. The default is `true`.
- `logRequestBody` (boolean)— Whether to log request bodies along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.
- `logResponseBody` (boolean)— Whether to log response bodies along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.
- `logRequestHeaders` (boolean)— Whether to log request headers along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.

Note that `includeRequestHeaders` enables headers in artifacts.

- `logResponseHeaders` (boolean)— Whether to log response headers along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.

Note that `includeResponseHeaders` enables headers in artifacts.

Note

The `Duration` and `SuccessPercent` metrics are always emitted for each canary, both with and without the `CanaryName` metric.

Methods to enable or disable metrics

disableAggregatedRequestMetrics()

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

disableRequestMetrics()

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

disableStepMetrics()

Disables all step metrics, including both step success metrics and step duration metrics.

enableAggregatedRequestMetrics()

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

enableRequestMetrics()

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

enableStepMetrics()

Enables all step metrics, including both step success metrics and step duration metrics.

get2xxMetric()

Returns whether the canary emits a `2xx` metric with the `CanaryName` dimension.

get4xxMetric()

Returns whether the canary emits a `4xx` metric with the `CanaryName` dimension.

get5xxMetric()

Returns whether the canary emits a `5xx` metric with the `CanaryName` dimension.

getAggregated2xxMetric()

Returns whether the canary emits a `2xx` metric with no dimension.

getAggregated4xxMetric()

Returns whether the canary emits a `4xx` metric with no dimension.

getAggregatedFailedCanaryMetric()

Returns whether the canary emits a `Failed` metric with no dimension.

getAggregatedFailedRequestsMetric()

Returns whether the canary emits a `Failed requests` metric with no dimension.

getAggregated5xxMetric()

Returns whether the canary emits a `5xx` metric with no dimension.

getFailedCanaryMetric()

Returns whether the canary emits a `Failed` metric with the `CanaryName` dimension.

getFailedRequestsMetric()

Returns whether the canary emits a `Failed requests` metric with the `CanaryName` dimension.

getStepDurationMetric()

Returns whether the canary emits a `Duration` metric with the `CanaryName` dimension for this canary.

getStepSuccessMetric()

Returns whether the canary emits a `StepSuccess` metric with the `CanaryName` dimension for this canary.

with2xxMetric(_2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with the `CanaryName` dimension for this canary.

with4xxMetric(_4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with the `CanaryName` dimension for this canary.

with5xxMetric(_5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(aggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with no dimension for this canary.

withAggregated4xxMetric(aggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with no dimension for this canary.

withAggregated5xxMetric(aggregated5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with no dimension for this canary.

withAggregatedFailedCanaryMetric(aggregatedFailedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

withAggregatedFailedRequestsMetric(aggregatedFailedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

withFailedCanaryMetric(failedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

withFailedRequestsMetric(failedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

withStepDurationMetric(stepDurationMetric)

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

withStepSuccessMetric(stepSuccessMetric)

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable other features

withHarFile()

Accepts a Boolean argument, which specifies whether to create a HAR file for this canary.

withStepsReport()

Accepts a Boolean argument, which specifies whether to report a step execution summary for this canary.

withIncludeUrlPassword()

Accepts a Boolean argument, which specifies whether to include passwords that appear in URLs in logs and reports.

withRestrictedUrlParameters()

Accepts an array of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. You can pass an asterisk (*) as a value to redact all URL path and query parameter values

withLogRequest()

Accepts a Boolean argument, which specifies whether to log every request in the canary's logs.

withLogResponse()

Accepts a Boolean argument, which specifies whether to log every response in the canary's logs.

withLogRequestBody()

Accepts a Boolean argument, which specifies whether to log every request body in the canary's logs.

withLogResponseBody()

Accepts a Boolean argument, which specifies whether to log every response body in the canary's logs.

withLogRequestHeaders()

Accepts a Boolean argument, which specifies whether to log every request header in the canary's logs.

withLogResponseHeaders()

Accepts a Boolean argument, which specifies whether to log every response header in the canary's logs.

getHarFile()

Returns whether the canary creates a HAR file.

getStepsReport()

Returns whether the canary reports a step execution summary.

getIncludeUrlPassword()

Returns whether the canary includes passwords that appear in URLs in logs and reports.

getRestrictedUrlParameters()

Returns whether the canary redacts URL path or query parameters.

getLogRequest()

Returns whether the canary logs every request in the canary's logs.

getLogResponse()

Returns whether the canary logs every response in the canary's logs.

getLogRequestBody()

Returns whether the canary logs every request body in the canary's logs.

getLogResponseBody()

Returns whether the canary logs every response body in the canary's logs.

getLogRequestHeaders()

Returns whether the canary logs every request header in the canary's logs.

getLogResponseHeaders()

Returns whether the canary logs every response header in the canary's logs.

Functions for all canaries

- `withIncludeRequestHeaders(includeRequestHeaders)`
- `withIncludeResponseHeaders(includeResponseHeaders)`
- `withRestrictedHeaders(restrictedHeaders)`
- `withIncludeRequestBody(includeRequestBody)`
- `withIncludeResponseBody(includeResponseBody)`
- `enableReportingOptions()`— Enables all reporting options-- `includeRequestHeaders`, `includeResponseHeaders`, `includeRequestBody`, and `includeResponseBody` .
- `disableReportingOptions()`— Disables all reporting options-- `includeRequestHeaders`, `includeResponseHeaders`, `includeRequestBody`, and `includeResponseBody` .

[setConfig\(options\) for UI canaries](#)

For UI canaries, `setConfig` can include the following Boolean parameters:

- `continueOnStepFailure (boolean)`— Whether to continue with running the canary script after a step fails (this refers to the `executeStep` function). If any steps fail, the canary run will still be marked as failed. The default is `false`.
- `harFile (boolean)`— Whether to create a HAR file. The default is `True`.
- `screenshotOnStepStart (boolean)`— Whether to take a screenshot before starting a step.
- `screenshotOnStepSuccess (boolean)`— Whether to take a screenshot after completing a successful step.
- `screenshotOnStepFailure (boolean)`— Whether to take a screenshot after a step fails.

[Methods to enable or disable screenshots](#)

disableStepScreenshots()

Disables all screenshot options (`screenshotOnStepStart`, `screenshotOnStepSuccess`, and `screenshotOnStepFailure`).

enableStepScreenshots()

Enables all screenshot options (`screenshotOnStepStart`, `screenshotOnStepSuccess`, and `screenshotOnStepFailure`). By default, all these methods are enabled.

getScreenshotOnStepFailure()

Returns whether the canary takes a screenshot after a step fails.

getScreenshotOnStepStart()

Returns whether the canary takes a screenshot before starting a step.

getScreenshotOnStepSuccess()

Returns whether the canary takes a screenshot after completing a step successfully.

withScreenshotOnStepStart(screenshotOnStepStart)

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

withScreenshotOnStepSuccess(screenshotOnStepSuccess)

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

withScreenshotOnStepFailure(screenshotOnStepFailure)

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

Usage in UI canaries

First, import the `Synthetics` dependency and fetch the configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();
```

Then, set the configuration for each option by calling the `setConfig` method using one of the following options.

```
// Set configuration values
synConfig.setConfig({
    screenshotOnStepStart: true,
    screenshotOnStepSuccess: false,
    screenshotOnStepFailure: false
});
```

Or

```
synConfig.withScreenshotOnStepStart(false).withScreenshotOnStepSuccess(true).withScreenshotOnStepFailure(false);
```

To disable all screenshots, use the `disableStepScreenshots()` function as in this example.

```
synConfig.disableStepScreenshots();
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

[setConfig\(options\) for API canaries](#)

For API canaries, `setConfig` can include the following Boolean parameters:

- `continueOnHttpStepFailure` (boolean)— Whether to continue with running the canary script after an HTTP step fails (this refers to the `executeHttpStep` function). If any steps fail, the canary run will still be marked as failed. The default is `true`.

Visual monitoring

Visual monitoring compares screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails and you can see the areas with differences highlighted in color in the canary run report. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium.

To enable visual monitoring, add the following line of code to the canary script. For more details, see [SyntheticsConfiguration class \(p. 201\)](#).

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see [Editing or deleting a canary \(p. 246\)](#).

Other options for visual monitoring

syntheticsConfiguration.withVisualVarianceThresholdPercentage(desiredPercentage)

Set the acceptable percentage for screenshot variance in visual comparisons.

syntheticsConfiguration.withVisualVarianceHighlightHexColor("#fafafa00")

Set the highlight color that designates variance areas when you look at canary run reports that use visual monitoring.

syntheticsConfiguration.withFail CanaryRunOnVisualVariance(fail Canary)

Set whether or not the canary fails when there is a visual difference that is more than the threshold. The default is to fail the canary.

Synthetics logger

SyntheticsLogger writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

You can use the SyntheticsLogger, assuming you want to run the Synthetics Library at the same log level as your Synthetics canary logging.

Using the SyntheticsLogger is not required to create a log file that is uploaded to your S3 results location. You could instead create a different log file in the /tmp folder. Any files created under the /tmp folder are uploaded to the results location in S3 as artifacts.

To use the Synthetics Library logger:

```
const log = require('SyntheticsLogger');
```

Useful function definitions:

log.debug(*message, ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.debug("Starting step - login.");
```

log.error(*message, ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
try {
  await login();
} catch (ex) {
  log.error("Error encountered in step - login.", ex);
}
```

log.info(*message, ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.info("Successfully completed step - login.");
```

log.log(*message, ex*);

This is an alias for `log.info`.

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.log("Successfully completed step - login.");
```

log.warn(*message, ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.warn("Exception encountered trying to publish CloudWatch Metric.", ex);
```

SyntheticsLogHelper class

The `SyntheticsLogHelper` class is available in the runtime `syn-nodejs-puppeteer-3.2` and later runtimes. It is already initialized in the CloudWatch Synthetics library and is configured with Synthetics

configuration. You can add this as a dependency in your script. This class enables you to sanitize URLs, headers, and error messages to redact sensitive information.

Note

Synthetics sanitizes all URLs and error messages it logs before including them in logs, reports, HAR files, and canary run errors based on the Synthetics configuration setting `restrictedUrlParameters`. You have to use `getSanitizedUrl` or `getSanitizedErrorMessage` only if you are logging URLs or errors in your script. Synthetics does not store any canary artifacts except for canary errors thrown by the script. Canary run artifacts are stored in your customer account. For more information, see [Security considerations for Synthetics canaries \(p. 955\)](#).

getSanitizedUrl(url, stepConfig = null)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized url strings based on the configuration. You can choose to redact sensitive URL parameters such as password and `access_token` by setting the property `restrictedUrlParameters`. By default, passwords in URLs are redacted. You can enable URL passwords if needed by setting `includeUrlPassword` to true.

This function throws an error if the URL passed in is not a valid URL.

Parameters

- `url` is a string and is the URL to sanitize.
- `stepConfig` (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following sample URL: `https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`. In this example, `access_token` contains your sensitive information which shouldn't be logged. Note that the Synthetics services doesn't store any canary run artifacts. Artifacts such as logs, screenshots, and reports are all stored in an Amazon S3 bucket in your customer account.

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
  restrictedUrlParameters: ['access_token'];
});
```

Next, sanitize and log the URL

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200');
```

This logs the following in your canary log.

```
My example url is: https://example.com/learn/home?  
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

You can override the Synthetics configuration for a URL by passing in an optional parameter containing Synthetics configuration options, as in the following example.

```
const urlConfig = {  
    restrictedUrlParameters: ['*']  
};  
const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?  
access_token=12345&token_type=Bearer&expires_in=1200', urlConfig);  
logger.info('My example url is: ' + sanitizedUrl);
```

The preceding example redacts all query parameters, and is logged as follows:

```
My example url is: https://example.com/learn/home?  
access_token=REDACTED&token_type=REDACTED&expires_in=REDACTED
```

getSanitizedErrorMessage

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized error strings by sanitizing any URLs present based on the Synthetics configuration. You can choose to override the global Synthetics configuration when you call this function by passing an optional `stepConfig` parameter.

Parameters

- `error` is the error to sanitize. It can be an Error object or a string.
- `stepConfig` (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following error: Failed to load url: https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency  
const synthetics = require('Synthetics');  
  
// Import Synthetics logger for logging url  
const log = require('SyntheticsLogger');  
  
// Get Synthetics configuration  
const synConfig = synthetics.getConfiguration();  
  
// Set restricted parameters  
synConfig.setConfig({  
    restrictedUrlParameters: ['access_token'];  
});
```

Next, sanitize and log the error message

```
// Import SyntheticsLogHelper dependency  
const syntheticsLogHelper = require('SyntheticsLogHelper');
```

```
try {
    // Your code which can throw an error containing url which your script logs
} catch (error) {
    const sanitizedErrorMessage = synthetics.getSanitizedErrorMessage(errorMessage);
    logger.info(sanitizedErrorMessage);
}
```

This logs the following in your canary log.

```
Failed to load url: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

[getSanitizedHeaders\(headers, stepConfig=null\)](#)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized headers based on the `restrictedHeaders` property of `syntheticsConfiguration`. The headers specified in the `restrictedHeaders` property are redacted from logs, HAR files, and reports.

Parameters

- `headers` is an object containing the headers to sanitize.
- `stepConfig` (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the headers.

[Node.js library classes and functions that apply to UI canaries only](#)

The following CloudWatch Synthetics library functions for Node.js are useful only for UI canaries.

Topics

- [Synthetics class \(p. 213\)](#)
- [BrokenLinkCheckerReport class \(p. 218\)](#)
- [SyntheticsLink class \(p. 219\)](#)

[Synthetics class](#)

The following functions are in the `Synthetics` class.

```
async addUserAgent(page, userAgentString);
```

This function appends `userAgentString` to the specified page's user-agent header.

Example:

```
await synthetics.addUserAgent(page, "MyApp-1.0");
```

Results in the page's user-agent header being set to `browsers-user-agent-header-valueMyApp-1.0`

```
async executeStep(stepName, functionToExecute, [stepConfig]);
```

Executes the provided step, wrapping it with start/pass/fail logging, start/pass/fail screenshots, and pass/fail and duration metrics.

Note

If you are using the `syn-nodejs-2.1` or later runtime, you can configure whether and when screenshots are taken. For more information, see [SyntheticsConfiguration class \(p. 201\)](#).

The `executeStep` function also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.
- Emits the `stepName SuccessPercent` metric, 100 for pass or 0 for failure.
- Emits the `stepName Duration` metric, with a value based on the step start and end times.
- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

If the canary uses the `syn-nodejs-2.0` runtime or later, this function also adds a step execution summary to the canary's report. The summary includes details about each step, such as start time, end time, status (PASSED/FAILED), failure reason (if failed), and screenshots captured during the execution of each step.

Example:

```
await synthetics.executeStep('navigateToUrl', async function (timeoutInMillis = 30000) {  
    await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:  
        timeoutInMillis});});
```

Response:

Returns what `functionToExecute` returns.

Updates with syn-nodejs-2.2

Starting with `syn-nodejs-2.2`, you can optionally pass step configurations to override CloudWatch Synthetics configurations at the step level. For a list of options that you can pass to `executeStep`, see [SyntheticsConfiguration class \(p. 201\)](#).

The following example overrides the default `false` configuration for `continueOnStepFailure` to `true` and specifies when to take screenshots.

```
var stepConfig = {  
    'continueOnStepFailure': true,  
    'screenshotOnStepStart': false,  
    'screenshotOnStepSuccess': true,  
    'screenshotOnStepFailure': false  
}  
  
await executeStep('Navigate to amazon', async function (timeoutInMillis = 30000) {  
    await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:  
        timeoutInMillis});  
}, stepConfig);
```

[getDefaultLaunchOptions\(\)](#)

The `getDefaultsLaunchOptions()` function returns the browser launch options that are used by CloudWatch Synthetics. For more information, see [puppeteer.launch\(\[options\]\)](#)

```
// This function returns default launch options used by Synthetics.  
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

getPage();

Returns the current open page as a Puppeteer object. For more information, see [Puppeteer API v1.14.0](#).

Example:

```
let page = synthetics.getPage();
```

Response:

The page (Puppeteer object) that is currently open in the current browser session.

getRequestResponseLogHelper();

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class \(p. 216\)](#) instead.

Use this function as a builder pattern for tweaking the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper().withLogRequestHeaders(false));
```

Response:

```
{RequestResponseLogHelper}
```

launch(options)

The options for this function are available only in the `syn-nodejs-2.1` runtime version or later.

This function is used only for UI canaries. It closes the existing browser and launches a new one.

Note

CloudWatch Synthetics always launches a browser before starting to run your script. You don't need to call `launch()` unless you want to launch a new browser with custom options.

(options) is a configurable set of options to set on the browser. For more information, see [puppeteer.launch\(\[options\]\)](#).

If you call this function with no options, Synthetics launches a browser with default arguments, `executablePath`, and `defaultViewport`. The default viewport in CloudWatch Synthetics is 1920 by 1080.

You can override launch parameters used by CloudWatch Synthetics and pass additional parameters when launching the browser. For example, the following code snippet launches a browser with default arguments and a default executable path, but with a viewport of 800 x 600.

```
await synthetics.launch({  
    defaultViewport: {
```

```
        "deviceScaleFactor": 1,  
        "width": 800,  
        "height": 600  
    }));
```

The following sample code adds a new `ignoreHTTPSErrors` parameter to the CloudWatch Synthetics launch parameters:

```
await synthetics.launch({  
    ignoreHTTPSErrors: true  
});
```

You can disable web security by adding a `--disable-web-security` flag to args in the CloudWatch Synthetics launch parameters:

```
// This function adds the --disable-web-security flag to the launch parameters  
const defaultOptions = await synthetics.getDefaultLaunchOptions();  
const launchArgs = [...defaultOptions.args, '--disable-web-security'];  
await synthetics.launch({  
    args: launchArgs  
});
```

RequestResponseLogHelper class

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this class is deprecated. Any use of this class causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class \(p. 216\)](#) instead.

Handles the fine-grained configuration and creation of string representations of request and response payloads.

```
class RequestResponseLogHelper {  
  
    constructor () {  
        this.request = {url: true, resourceType: false, method: false, headers: false,  
        postData: false};  
        this.response = {status: true, statusText: true, url: true, remoteAddress: false,  
        headers: false};  
    }  
  
    withLogRequestUrl(logRequestUrl);  
  
    withLogRequestResourceType(logRequestResourceType);  
  
    withLogRequestMethod(logRequestMethod);  
  
    withLogRequestHeaders(logRequestHeaders);  
  
    withLogRequestpostData(logRequestpostData);  
  
    withLogResponseStatus(logResponseStatus);  
  
    withLogResponseStatusText(logResponseStatusText);  
  
    withLogResponseUrl(logResponseUrl);  
  
    withLogResponseRemoteAddress(logResponseRemoteAddress);
```

```
withLogResponseHeaders(logResponseHeaders);
```

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper()  
.withLogRequestpostData(true)  
.withLogRequestHeaders(true)  
.withLogResponseHeaders(true));
```

Response:

```
{RequestResponseLogHelper}
```

[setRequestResponseLogHelper\(\)](#);

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class \(p. 216\)](#) instead.

Use this function as a builder pattern for setting the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper().withLogRequestHeaders(true).withLogResponseHeaders(true);
```

Response:

```
{RequestResponseLogHelper}
```

[async takeScreenshot\(name, suffix\);](#)

Takes a screenshot (.PNG) of the current page with name and suffix (optional).

Example:

```
await synthetics.takeScreenshot("navigateToUrl", "loaded")
```

This example captures and uploads a screenshot named `01-navigateToUrl-loaded.png` to the canary's S3 bucket.

You can take a screenshot for a particular canary step by passing the `stepName` as the first parameter. Screenshots are linked to the canary step in your reports, to help you track each step while debugging.

CloudWatch Synthetics canaries automatically take screenshots before starting a step (the `executeStep` function) and after the step completion (unless you configure the canary to disable screenshots). You can take more screenshots by passing in the step name in the `takeScreenshot` function.

The following example takes screenshot with the `signupForm` as the value of the `stepName`. The screenshot will be named `02-signupForm-address` and will be linked to the step named `signupForm` in the canary report.

```
await synthetics.takeScreenshot('signupForm', 'address')
```

BrokenLinkCheckerReport class

This class provides methods to add a synthetics link. It's supported only on canaries that use the syn-nodejs-2.0-beta version of the runtime or later.

To use `BrokenLinkCheckerReport`, include the following lines in the script:

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

const brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

Useful function definitions:

`addLink(syntheticsLink, isBroken)`

syntheticsLink is a `SyntheticsLink` object representing a link. This function adds the link according to the status code. By default, it considers a link to be broken if the status code is not available or the status code is 400 or higher. You can override this default behavior by passing in the optional parameter `isBrokenLink` with a value of `true` or `false`.

This function does not have a return value.

`getLinks()`

This function returns an array of `SyntheticsLink` objects that are included in the broken link checker report.

`getTotalBrokenLinks()`

This function returns a number representing the total number of broken links.

`getTotalLinksChecked()`

This function returns a number representing the total number of links included in the report.

How to use `BrokenLinkCheckerReport`

The following canary script code snippet demonstrates an example of navigating to a link and adding it to the broken link checker report.

1. Import `SyntheticsLink`, `BrokenLinkCheckerReport`, and `Synthetics`.

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');
const SyntheticsLink = require('SyntheticsLink');

// Synthetics dependency
const synthetics = require('Synthetics');
```

2. To add a link to the report, create an instance of `BrokenLinkCheckerReport`.

```
let brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

3. Navigate to the URL and add it to the broken link checker report.

```
let url = "https://amazon.com";

let syntheticsLink = new SyntheticsLink(url);

// Navigate to the url.
let page = await synthetics.getPage();
```

```
// Create a new instance of Synthetics Link
let link = new SyntheticsLink(url)

try {
    const response = await page.goto(url, {waitUntil: 'domcontentloaded', timeout: 30000});
} catch (ex) {
    // Add failure reason if navigation fails.
    link.withFailureReason(ex);
}

if (response) {
    // Capture screenshot of destination page
    let screenshotResult = await synthetics.takeScreenshot('amazon-home', 'loaded');

    // Add screenshot result to synthetics link
    link.addScreenshotResult(screenshotResult);

    // Add status code and status description to the link
    link.withStatusCode(response.status()).withStatusText(response.statusText())
}

// Add link to broken link checker report.
brokenLinkCheckerReport.addLink(link);
```

4. Add the report to Synthetics. This creates a JSON file named `BrokenLinkCheckerReport.json` in your S3 bucket for each canary run. You can see a links report in the console for each canary run along with screenshots, logs, and HAR files.

```
await synthetics.addReport(brokenLinkCheckerReport);
```

SyntheticsLink class

This class provides methods to wrap information. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `SyntheticsLink`, include the following lines in the script:

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

const syntheticsLink = new SyntheticsLink("https://www.amazon.com");
```

This function returns `syntheticsLinkObject`

Useful function definitions:

withUrl(*url*)

url is a URL string. This function returns `syntheticsLinkObject`

withText(*text*)

text is a string representing anchor text. This function returns `syntheticsLinkObject`. It adds anchor text corresponding to the link.

withParentUrl(*parentUrl*)

parentUrl is a string representing the parent (source page) URL. This function returns `syntheticsLinkObject`

withStatusCode(*statusCode*)

`statusCode` is a string representing the status code. This function returns `syntheticsLinkObject`

withFailureReason(`failureReason`)

`failureReason` is a string representing the failure reason. This function returns `syntheticsLinkObject`

addScreenshotResult(`screenshotResult`)

`screenshotResult` is an object. It is an instance of `ScreenshotResult` that was returned by the Synthetics function `takeScreenshot`. The object includes the following:

- `fileName`—A string representing the `screenshotFileName`
- `pageUrl` (optional)
- `error` (optional)

Node.js library classes and functions that apply to API canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for API canaries.

Topics

- [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\) \(p. 220\)](#)

[executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\)](#)

Executes the provided HTTP request as a step, and publishes `SuccessPercent` (pass/fail) and `Duration` metrics.

`executeHttpStep` uses either HTTP or HTTPS native functions under the hood, depending upon the protocol specified in the request.

This function also adds a step execution summary to the canary's report. The summary includes details about each HTTP request, such as the following:

- Start time
- End time
- Status (PASSED/FAILED)
- Failure reason, if it failed
- HTTP call details such as request/response headers, body, status code, status message, and performance timings.

Parameters

stepName(`String`)

Specifies the name of the step. This name is also used for publishing CloudWatch metrics for this step.

requestOptions(`Object or String`)

The value of this parameter can be a URL, a URL string, or an object. If it is an object, then it must be a set of configurable options to make an HTTP request. It supports all options in [`http.request\(options\[, callback\]\)`](#) in the Node.js documentation.

In addition to these Node.js options, `requestOptions` supports the additional parameter `body`. You can use the `body` parameter to pass data as a request body.

callback(**response**)

(Optional) This is a user function which is invoked with the HTTP response. The response is of the type [Class: http.IncomingMessage](#).

stepConfig(**object**)

(Optional) Use this parameter to override global synthetics configurations with a different configuration for this step.

Examples of using executeHttpStep

The following series of examples build on each other to illustrate the various uses of this option.

This first example configures request parameters. You can pass a URL as **requestOptions**:

```
let requestOptions = 'https://www.amazon.com';
```

Or you can pass a set of options:

```
let requestOptions = {
    'hostname': 'myproductsEndpoint.com',
    'method': 'GET',
    'path': '/test/product/validProductName',
    'port': 443,
    'protocol': 'https:'
};
```

The next example creates a callback function which accepts a response. By default, if you do not specify **callback**, CloudWatch Synthetics validates that the status is between 200 and 299 inclusive.

```
// Handle validation for positive scenario
const callback = async function(res) {
    return new Promise((resolve, reject) => {
        if (res.statusCode < 200 || res.statusCode > 299) {
            throw res.statusCode + ' ' + res.statusMessage;
        }

        let responseBody = '';
        res.on('data', (d) => {
            responseBody += d;
        });

        res.on('end', () => {
            // Add validation on 'responseBody' here if required. For ex, your status
            code is 200 but data might be empty
            resolve();
        });
    });
};
```

The next example creates a configuration for this step that overrides the global CloudWatch Synthetics configuration. The step configuration in this example allows request headers, response headers, request body (post data), and response body in your report and restrict 'X-Amz-Security-Token' and 'Authorization' header values. By default, these values are not included in the report for security reasons. If you choose to include them, the data is only stored in your S3 bucket.

```
// By default headers, post data, and response body are not included in the report for
// security reasons.
```

```
// Change the configuration at global level or add as step configuration for individual
// steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
    values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};
```

This final example passes your request to `executeHttpStep` and names the step.

```
await synthetics.executeHttpStep('Verify GET products API', requestOptions, callback,
    stepConfig);
```

With this set of examples, CloudWatch Synthetics adds the details from each step in your report and produces metrics for each step using `stepName`.

You will see `successPercent` and `duration` metrics for the `Verify GET products API` step. You can monitor your API performance by monitoring the metrics for your API call steps.

For a sample complete script that uses these functions, see [Multi-step API canary \(p. 237\)](#).

Library functions available for Python canary scripts using Selenium

This section lists the Selenium library functions available for Python canary scripts.

Topics

- [Python and Selenium library classes and functions that apply to all canaries \(p. 222\)](#)
- [Python and Selenium library classes and functions that apply to UI canaries only \(p. 228\)](#)

Python and Selenium library classes and functions that apply to all canaries

The following CloudWatch Synthetics Selenium library functions for Python are useful for all canaries.

Topics

- [SyntheticsConfiguration class \(p. 222\)](#)
- [SyntheticsLogger class \(p. 226\)](#)

SyntheticsConfiguration class

You can use the `SyntheticsConfiguration` class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level.

Function definitions:

`set_config(options)`

```
from aws_synthetics.common import synthetics_configuration
```

`options` is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in `options`.

- `screenshot_on_step_start` (boolean)— Whether to take a screenshot before starting a step.
- `screenshot_on_step_success` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshot_on_step_failure` (boolean)— Whether to take a screenshot after a step fails.

`withScreenshotOnStepStart(screenshotOnStepStart)`

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

`withScreenshotOnStepSuccess(screenshotOnStepSuccess)`

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

`withScreenshotOnStepFailure(screenshotOnStepFailure)`

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

`getScreenshotOnStepStart()`

Returns whether to take a screenshot before starting a step.

`getScreenshotOnStepSuccess()`

Returns whether to take a screenshot after completing a step successfully.

`getScreenshotOnStepFailure()`

Returns whether to take a screenshot after a step fails.

`disableStepScreenshots()`

Disables all screenshot options (`getScreenshotOnStepStart`, `getScreenshotOnStepSuccess`, and `getScreenshotOnStepFailure`).

`enableStepScreenshots()`

Enables all screenshot options (`getScreenshotOnStepStart`, `getScreenshotOnStepSuccess`, and `getScreenshotOnStepFailure`). By default, all these methods are enabled.

`setConfig(options) regarding CloudWatch metrics`

For canaries using `syn-python-selenium-1.1` or later, the **(options)** for `setConfig` can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see [CloudWatch metrics published by canaries \(p. 244\)](#).

- `failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failed_requests_metric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.

- `2xx_metric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `4xx_metric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `5xx_metric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `step_duration_metric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.
- `step_success_metric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.
- `aggregated_failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_failed_requests_metric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_2xx_metric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_4xx_metric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_5xx_metric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

with_2xx_metric(2xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with the `CanaryName` dimension for this canary.

with_4xx_metric(4xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with the `CanaryName` dimension for this canary.

with_5xx_metric(5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(aggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with no dimension for this canary.

withAggregated4xxMetric(aggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with no dimension for this canary.

with_aggregated_5xx_metric(aggregated_5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with no dimension for this canary.

with_aggregated_failed_canary_metric(aggregated_failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

with_aggregated_failed_requests_metric(aggregated_failed_requests_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

`with_failed_canary_metric(failed_canary_metric)`

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

`with_failed_requests_metric(failed_requests_metric)`

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

`with_step_duration_metric(step_duration_metric)`

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

`with_step_success_metric(step_success_metric)`

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable metrics

`disable_aggregated_request_metrics()`

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

`disable_request_metrics()`

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

`disable_step_metrics()`

Disables all step metrics, including both step success metrics and step duration metrics.

`enable_aggregated_request_metrics()`

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

`enable_request_metrics()`

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

`enable_step_metrics()`

Enables all step metrics, including both step success metrics and step duration metrics.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration. Then, set the configuration for each option by calling the `setConfig` method using one of the following options.

```
from aws_synthetics.common import synthetics_configuration
synthetics_configuration.set_config(
{
    "screenshot_on_step_start": False,
    "screenshot_on_step_success": False,
```

```
        "screenshot_on_step_failure": True
    }
)
or
```

Or

```
synthetics_configuration.with_screenshot_on_step_start(False).with_screenshot_on_step_success(False).wi
```

To disable all screenshots, use the `disableStepScreenshots()` function as in this example.

```
synthetics_configuration.disable_step/screenshots()
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

[set_config\(options\) for UI canaries](#)

Starting with `syn-python-selenium-1.1`, for UI canaries, `set_config` can include the following Boolean parameters:

- `continue_on_step_failure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the `executeStep` function). If any steps fail, the canary run will still be marked as failed. The default is `false`.

[SyntheticsLogger class](#)

`synthetics_logger` writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

Using `synthetics_logger` is not required to create a log file that is uploaded to your Amazon S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in the S3 bucket as artifacts.

To use `synthetics_logger`:

```
from aws_synthetics.common import synthetics_logger
```

Useful function definitions:

Get log level:

```
log_level = synthetics_logger.get_level()
```

Set log level:

```
synthetics_logger.set_level()
```

Log a message with a specified level. The level can be `DEBUG`, `INFO`, `WARN`, or `ERROR`, as in the following syntax examples:

```
synthetics_logger.debug(message, *args, **kwargs)
```

```
synthetics_logger.info(message, *args, **kwargs)
```

```
synthetics_logger.log(message, *args, **kwargs)
```

```
synthetics_logger.warn(message, *args, **kwargs)
```

```
synthetics_logger.error(message, *args, **kwargs)
```

For information about debug parameters, see the standard Python documentation at [logging.debug](#)

In these logging functions, the `message` is the message format string. The `args` are the arguments that are merged into `msg` using the string formatting operator.

There are three keyword arguments in `kwargs`:

- `exc_info`— If not evaluated as false, adds exception information to the logging message.
- `stack_info`— defaults to false. If true, adds stack information to the logging message, including the actual logging call.
- `extra`— The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the `LogRecord` created for the logging event with user-defined attributes.

Examples:

Log a message with the level DEBUG:

```
synthetics_logger.debug('Starting step - login.')
```

Log a message with the level INFO. `logger.log` is a synonym for `logger.info`:

```
synthetics_logger.info('Successfully completed step - login.')
```

or

```
synthetics_logger.log('Successfully completed step - login.')
```

Log a message with the level WARN:

```
synthetics_logger.warn('Warning encountered trying to publish %s', 'CloudWatch Metric')
```

Log a message with the level ERROR:

```
synthetics_logger.error('Error encountered trying to publish %s', 'CloudWatch Metric')
```

Log an exception:

```
synthetics_logger.exception(message, *args, **kwargs)
```

Logs a message with level `ERROR`. Exception information is added to the logging message. You should call this function only from an exception handler.

For information about exception parameters, see the standard Python documentation at [logging.exception](#)

The `message` is the message format string. The `args` are the arguments, which are merged into `msg` using the string formatting operator.

There are three keyword arguments in `kwargs`:

- `exc_info`— If not evaluated as `false`, adds exception information to the logging message.
- `stack_info`— defaults to `false`. If `true`, adds stack information to the logging message, including the actual logging call.
- `extra`— The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the `LogRecord` created for the logging event with user-defined attributes.

Example:

```
synthetics_logger.exception('Error encountered trying to publish %s', 'CloudWatch Metric')
```

Python and Selenium library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics Selenium library functions for Python are useful only for UI canaries.

Topics

- [SyntheticsBrowser class \(p. 228\)](#)
- [SyntheticsWebDriver class \(p. 229\)](#)

SyntheticsBrowser class

When you create a browser instance by calling `synthetics_webdriver.Chrome()`, the returned browser instance is of the type `SyntheticsBrowser`. The `SyntheticsBrowser` class controls the `ChromeDriver`, and enables the canary script to drive the browser, allowing the Selenium WebDriver to work with Synthetics.

In addition to the standard Selenium methods, it also provides the following methods.

`set_viewport_size(width, height)`

Sets the viewport of the browser. Example:

```
browser.set_viewport_size(1920, 1080)
```

`save_screenshot(filename, suffix)`

Saves screenshots to the `/tmp` directory. The screenshots are uploaded from there to the canary artifacts folder in the S3 bucket.

`filename` is the file name for the screenshot, and `suffix` is an optional string to be used for naming the screenshot.

Example:

```
browser.save_screenshot('loaded.png', 'page1')
```

SyntheticsWebDriver class

To use this class, use the following in your script:

```
from aws_synthetics.selenium import synthetics_webdriver
```

```
add_execution_error(errorMessage, ex);
```

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `add_execution_error` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `add_execution_error` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try:  
    browser.save_screenshot("loaded.png")  
except Exception as ex:  
    self.add_execution_error("Unable to take screenshot", ex)
```

add_user_agent(user_agent_str)

Appends the value of `user_agent_str` to the browser's user agent header. You must assign `user_agent_str` before creating the browser instance.

Example:

```
synthetics_webdriver.add_user_agent('MyApp-1.0')
```

execute_step(step_name, function_to_execute)

Processes one function. It also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.
- Emits the `stepName SuccessPercent` metric, 100 for pass or 0 for failure.
- Emits the `stepName Duration` metric, with a value based on the step start and end times.

- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

Example:

```
def custom_actions():
    #verify contains
    browser.find_element_by_xpath("//*[@id=\"id_1\"]*[contains(text(),'login')]")
    #click a button
    browser.find_element_by_xpath('//*[@id="submit"]/a').click()

    await synthetics_webdriver.execute_step("verify_click", custom_actions)
```

Chrome()

Launches an instance of the Chromium browser and returns the created instance of the browser.

Example:

```
browser = synthetics_webdriver.Chrome()
browser.get("https://example.com/")
```

To launch a browser in incognito mode, use the following:

```
add_argument('--incognito')
```

To add proxy settings, use the following:

```
add_argument('--proxy-server=%s' % PROXY)
```

Example:

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument("--incognito")
browser = syn_webdriver.Chrome(chrome_options=chrome_options)
```

Scheduling canary runs using cron

Using a cron expression gives you flexibility when you schedule a canary. Cron expressions contain five or six fields in the order listed in the following table. The fields are separated by spaces. The syntax differs depending on whether you are using the CloudWatch console to create the canary, or the AWS CLI or AWS SDKs. When you use the console, you specify only the first five fields. When you use the AWS CLI or AWS SDKs, you specify all six fields, and you must specify * for the Year field.

| Field | Allowed values | Allowed special characters |
|--------------|-----------------|----------------------------|
| Minutes | 0-59 | , - * / |
| Hours | 0-23 | , - * / |
| Day-of-month | 1-31 | , - * ? / L W |
| Month | 1-12 or JAN-DEC | , - * / |
| Day-of-week | 1-7 or SUN-SAT | , - * ? L # |

| Field | Allowed values | Allowed special characters |
|-------|----------------|----------------------------|
| Year | * | |

Special characters

- The , (comma) includes multiple values in the expression for a field. For example, in the Month field, JAN,FEB,MAR would include January, February, and March.
- The - (dash) special character specifies ranges. In the Day field, 1-15 would include days 1 through 15 of the specified month.
- The * (asterisk) special character includes all values in the field. In the Hours field, * includes every hour. You cannot use * in both the Day-of-month and Day-of-week fields in the same expression. If you use it in one, you must use ? in the other.
- The / (forward slash) specifies increments. In the Minutes field, you can enter 1/10 to specify every tenth minute, starting from the first minute of the hour (for example, the eleventh, twenty-first, and thirty-first minute, and so on).
- The ? (question mark) specifies one or another. If you enter 7 in the Day-of-month field and you don't care what day of the week the seventh is, you can enter ? in the Day-of-week field.
- The L wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week.
- The w wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, 3w specifies the weekday closest to the third day of the month.
- The # wildcard in the Day-of-week field specifies a certain instance of the specified day of the week within a month. For example, 3#2 is the second Tuesday of the month. The 3 refers to Tuesday because it is the third day of each week, and the 2 refers to the second day of that type within the month.

Limitations

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value or * (asterisk) in one of the fields, you must use a ? (question mark) in the other.
- Cron expressions that lead to rates faster than one minute are not supported.
- You can't set a canary to wait for more than a year before running, so you can specify only * in the Year field.

Examples

You can refer to the following sample cron strings when you create a canary. The following examples are the correct syntax for using the AWS CLI or AWS SDKs to create or update a canary. If you are using the CloudWatch console, omit the final * in each example.

| Expression | Meaning |
|------------------------|--|
| 0 10 * * ? * | Run at 10:00 am (UTC) every day |
| 15 12 * * ? * | Run at 12:15 am (UTC) every day |
| 0 18 ? * MON-FRI * | Run at 6:00 am (UTC) every Monday through Friday |
| 0 8 1 * ? * | Run at 8:00 am (UTC) on the first day of each month |
| 0/10 * ? * MON-SAT * | Run every 10 minutes Monday through Saturday of each week |
| 0/5 8-17 ? * MON-FRI * | Run every five minutes Monday through Friday between 8:00 am and 5:55 pm (UTC) |

Troubleshooting a failed canary

If your canary fails, check the following for troubleshooting.

General troubleshooting

- Use the canary details page to find more information. In the CloudWatch console, choose **Canaries** in the navigation pane and then choose the name of the canary to open the canary details page. In the **Availability** tab, check the **SuccessPercent** metric to see whether the problem is constant or intermittent.

While still in the **Availability** tab, choose a failed data point to see screenshots, logs, and step reports (if available) for that failed run.

If a step report is available because steps are part of your script, check to see which step has failed and see the associated screenshots to see the issue that your customers are seeing.

You can also check the HAR files to see if one or more requests are failing. You can dig deeper by using logs to drill down on failed requests and errors. Finally, you can compare these artifacts with the artifacts from a successful canary run to pinpoint the issue.

By default, CloudWatch Synthetics captures screenshots for each step in a UI canary. However, your script might be configured to disable screenshots. During debugging, you may want to enable screenshots again. Similarly, for API canaries you might want to see HTTP request and response headers and body during debugging. For information about how to include this data in the report, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\) \(p. 220\)](#).

- If you had a recent deployment to your application, roll it back and then debug later.
- Connect to your endpoint manually to see if you can reproduce the same issue.

Topics

- [Canary runtime version upgrade and downgrade issues \(p. 232\)](#)
- [Waiting for an element to appear \(p. 233\)](#)
- [Node is either not visible or not an HTMLElement for page.click\(\) \(p. 233\)](#)
- [Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied \(p. 233\)](#)
- [Error: Protocol error \(Runtime.callFunctionOn\): Target closed. \(p. 233\)](#)
- [Canary Failed. Error: No datapoint - Canary Shows timeout error \(p. 234\)](#)
- [Trying to access an internal endpoint \(p. 234\)](#)
- [Cross-origin request sharing \(CORS\) issue \(p. 234\)](#)
- [Troubleshooting a canary on a VPC \(p. 235\)](#)

Canary runtime version upgrade and downgrade issues

If you recently upgraded the canary from runtime version syn-1.0 to a later version, it may be a cross-origin request sharing (CORS) issue. For more information, see [Cross-origin request sharing \(CORS\) issue \(p. 234\)](#).

If you recently downgraded the canary to an older runtime version, check to make sure that the CloudWatch Synthetics functions that you are using are available in the older runtime version that you

downgraded to. For example, the `executeHttpStep` function is available for runtime version `syn-nodejs-2.2` and later. To check on the availability of functions, see [Writing a canary script \(p. 192\)](#).

Note

When you plan to upgrade or downgrade the runtime version for a canary, we recommend that you first clone the canary and update the runtime version in the cloned canary. Once you have verified that the clone with the new runtime version works, you can update the runtime version of your original canary and delete the clone.

Waiting for an element to appear

After analyzing your logs and screenshots, if you see that your script is waiting for an element to appear on screen and times out, check the relevant screenshot to see if the element appears on the page. Verify your `xpath` to make sure that it is correct.

For Puppeteer-related issues, check [Puppeteer's GitHub page](#) or internet forums.

Node is either not visible or not an HTMLElement for page.click()

If a node is not visible or is not an `HTMLElement` for `page.click()`, first verify the `xpath` that you are using to click the element. Also, if your element is at the bottom of the screen, adjust your viewport. CloudWatch Synthetics by default uses a viewport of `1920 * 1080`. You can set a different viewport when you launch the browser or by using the Puppeteer function `page.setViewport`.

Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied

If your canary fails because of an Amazon S3 error, CloudWatch Synthetics was unable to upload screenshots, logs, or reports created for the canary because of permission issues. Check the following:

- Check that the canary's IAM role has the `s3>ListAllMyBuckets` permission, the `s3:GetBucketLocation` permission for the correct Amazon S3 bucket, and the `s3:PutObject` permission for the bucket where the canary stores its artifacts. If the canary performs visual monitoring, the role also needs the `s3:GetObject` permission for the bucket.
- If the canary uses an AWS KMS customer managed key for encryption instead of the standard AWS managed key (default), the canary's IAM role might not have the permission to encrypt or decrypt using that key. For more information, see [Encrypting canary artifacts \(p. 241\)](#).
- Your bucket policy might not allow the encryption mechanism that the canary uses. For example, if your bucket policy mandates to use a specific encryption mechanism or KMS key, then you must select the same encryption mode for your canary.

If the canary performs visual monitoring, see [Updating artifact location and encryption when using visual monitoring \(p. 242\)](#) for more information.

Error: Protocol error (Runtime.callFunctionOn): Target closed.

This error appears if there are some network requests after the page or browser is closed. You might have forgotten to wait for an asynchronous operation. After executing your script, CloudWatch Synthetics closes the browser. The execution of any asynchronous operation after the browser is closed might cause `target closed` error.

Canary Failed. Error: No datapoint - Canary Shows timeout error

This means that your canary run exceeded the timeout. The canary execution stopped before CloudWatch Synthetics could publish success percent CloudWatch metrics or update artifacts such as HAR files, logs and screenshots. If your timeout is too low, you can increase it.

By default, a canary timeout value is equal to its frequency. You can manually adjust the timeout value to be less than or equal to the canary frequency. If your canary frequency is low, you must increase the frequency to increase the timeout. You can adjust both the frequency and the timeout value under **Schedule** when you create or update a canary by using the CloudWatch Synthetics console.

Be sure that your canary timeout value is no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

Canary artifacts are not available to view in the CloudWatch Synthetics console when this error happens. You can use CloudWatch Logs to see the canary's logs.

To use CloudWatch Logs to see the logs for a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Log groups**.
3. Find the log group by typing the canary name in the filter box. Log groups for canaries have the name `/aws/lambda/cwsyn-canaryName-randomId`.

Trying to access an internal endpoint

If you want your canary to access an endpoint on your internal network, we recommend that you set up CloudWatch Synthetics to use VPC. For more information, see [Running a canary on a VPC \(p. 240\)](#).

Cross-origin request sharing (CORS) issue

In a UI canary, if some network requests are failing with 403 or `net::ERR_FAILED`, check whether the canary has active tracing enabled and also uses the Puppeteer function `page.setExtraHTTPHeaders` to add headers. If so, the failed network requests might be caused by cross-origin request sharing (CORS) restrictions. You can confirm whether this is the case by disabling active tracing or removing the extra HTTP headers.

Why does this happen?

When active tracing is used, an extra header is added to all outgoing requests to trace the call. Modifying the request headers by adding a trace header or adding extra headers using Puppeteer's `page.setExtraHTTPHeaders` causes a CORS check for XMLHttpRequest (XHR) requests.

If you don't want to disable active tracing or remove the extra headers, you can update your web application to allow cross-origin access or you can disable web security by using the `disable-web-security` flag when you launch the Chrome browser in your script.

You can override launch parameters used by CloudWatch Synthetics and pass additional `disable-web-security` flag parameters by using the CloudWatch Synthetics launch function. For more information, see [Library functions available for Node.js canary scripts \(p. 200\)](#).

Note

You can override launch parameters used by CloudWatch Synthetics when you use runtime version `syn-nodejs-2.1` or later.

Troubleshooting a canary on a VPC

If you have issues after creating or updating a canary on a VPC, one of the following sections might help you troubleshoot the problem.

New canary in error state or canary can't be updated

If you create a canary to run on a VPC and it immediately goes into an error state, or you can't update a canary to run on a VPC, the canary's role might not have the right permissions. To run on a VPC, a canary must have the permissions `ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces`, and `ec2:DeleteNetworkInterface`. These permissions are all contained in the `AWSLambdaVPCAccessExecutionRole` managed policy. For more information, see [Execution Role and User Permissions](#).

If this issue happened when you created a canary, you must delete the canary, and create a new one. If you use the CloudWatch console to create the new canary, under **Access Permissions**, select **Create a new role**. A new role that includes all permissions required to run the canary is created.

If this issue happens when you update a canary, you can update the canary again and provide a new role that has the required permissions.

"No test result returned" error

If a canary displays a "no test result returned" error, one of the following issues might be the cause:

- If your VPC does not have internet access, you must use VPC endpoints to give the canary access to CloudWatch and Amazon S3. You must enable the **DNS resolution** and **DNS hostname** options in the VPC for these endpoint addresses to resolve correctly. For more information, see [Using DNS with Your VPC](#).
- Canaries must run in private subnets within a VPC. To check this, open the **Subnets** page in the VPC console. Check the subnets that you selected when configuring the canary. If they have a path to an internet gateway (`igw-`), they are not private subnets.

To help you troubleshoot these issues, see the logs for the canary.

To see the log events from a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the canary's log group. The log group name starts with `/aws/lambda/cwsyn-canary-name`.

Sample code for canary scripts

This section contains code samples that illustrate some possible functions for CloudWatch Synthetics canary scripts.

Samples for Node.js and Puppeteer

Setting cookies

Web sites rely on cookies to provide custom functionality or track users. By setting cookies in CloudWatch Synthetics scripts, you can mimic this custom behavior and validate it.

For example, a web site might display a **Login** link for a revisiting user instead of a **Register** link.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadBlueprint = async function () {

    let url = "http://smile.amazon.com/";

    let page = await synthetics.getPage();

    // Set cookies. I found that name, value, and either url or domain are required
    // fields.
    const cookies = [
        {
            'name': 'cookie1',
            'value': 'val1',
            'url': url
        },
        {
            'name': 'cookie2',
            'value': 'val2',
            'url': url
        },
        {
            'name': 'cookie3',
            'value': 'val3',
            'url': url
        }
    ];

    await page.setCookie(...cookies);

    // Navigate to the url
    await synthetics.executeStep('pageLoaded_home', async function (timeoutInMillis =
30000) {

        var response = await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
timeoutInMillis});

        // Log cookies for this page and this url
        const cookiesSet = await page.cookies(url);
        log.info("Cookies for url: " + url + " are set to: " + JSON.stringify(cookiesSet));
    });
};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Device emulation

You can write scripts that emulate various devices so that you can approximate how a page looks and behaves on those devices.

The following sample emulates an iPhone 6 device. For more information about emulation, see [page.emulate\(options\)](#) in the Puppeteer documentation.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');
const puppeteer = require('puppeteer-core');

const pageLoadBlueprint = async function () {

    const iPhone = puppeteer.devices['iPhone 6'];
```

```
// INSERT URL here
const URL = "https://amazon.com";

let page = await synthetics.getPage();
await page.emulate(iPhone);

//You can customize the wait condition here. For instance,
//using 'networkidle2' may be less restrictive.
const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout: 30000});
if (!response) {
    throw "Failed to load page!";
}

await page.waitFor(15000);

await synthetics.takeScreenshot('loaded', 'loaded');

//If the response status code is not a 2xx success code
if (response.status() < 200 || response.status() > 299) {
    throw "Failed to load page!";
};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Multi-step API canary

This sample code demonstrates an API canary with two HTTP steps: testing the same API for positive and negative test cases. The step configuration is passed to enable reporting of request/response headers. Additionally, it hides the Authorization header and X-Amz-Security-Token, because they contain user credentials.

When this script is used as a canary, you can view details about each step and the associated HTTP requests such as step pass/fail, duration, and performance metrics like DNS look up time and first byte time. You can view the number of 2xx, 4xx and 5xx for your canary run.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const apiCanaryBlueprint = async function () {

    // Handle validation for positive scenario
    const validatePositiveCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusText;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your status
                code is 200 but data might be empty
                resolve();
            });
        });
    };
};
```

```

// Handle validation for negative scenario
const validateNegativeCase = async function(res) {
    return new Promise((resolve, reject) => {
        if (res.statusCode < 400) {
            throw res.statusCode + ' ' + res.statusMessage;
        }

        resolve();
    });
};

let requestOptionsStep1 = {
    'hostname': 'myproductsEndpoint.com',
    'method': 'GET',
    'path': '/test/product/validProductName',
    'port': 443,
    'protocol': 'https:'
};

let headers = {};
headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-Agent']].join(' ');

requestOptionsStep1['headers'] = headers;

// By default headers, post data and response body are not included in the report for
// security reasons.
// Change the configuration at global level or add as step configuration for individual
steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};

await synthetics.executeHttpStep('Verify GET products API with valid name',
requestOptionsStep1, validatePositiveCase, stepConfig);

let requestOptionsStep2 = {
    'hostname': 'myproductsEndpoint.com',
    'method': 'GET',
    'path': '/test/canary/InvalidName(',
    'port': 443,
    'protocol': 'https:'
};

headers = {};
headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-Agent']].join(' ');

requestOptionsStep2['headers'] = headers;

// By default headers, post data and response body are not included in the report for
// security reasons.
// Change the configuration at global level or add as step configuration for individual
steps
stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
values do not appear in report generated.
}

```

```
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpStep('Verify GET products API with invalid name',
        requestOptionsStep2, validateNegativeCase, stepConfig);

};

exports.handler = async () => {
    return await apiCanaryBlueprint();
};
```

Samples for Python and Selenium

The following sample Selenium code is a canary that fails with a custom error message when a target element is not loaded.

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
from aws_synthetics.common import synthetics_logger as logger
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

def custom_selenium_script():
    # create a browser instance
    browser = webdriver.Chrome()
    browser.get('https://www.example.com/')
    logger.info('navigated to home page')
    # set cookie
    browser.add_cookie({'name': 'foo', 'value': 'bar'})
    browser.get('https://www.example.com/')
    # save screenshot
    browser.save_screenshot('signed.png')
    # expected status of an element
    button_condition = EC.element_to_be_clickable((By.CSS_SELECTOR, '.submit-button'))
    # add custom error message on failure
    WebDriverWait(browser, 5).until(button_condition, message='Submit button failed to
load').click()
    logger.info('Submit button loaded successfully')
    # browser will be quit automatically at the end of canary run,
    # quit action is not necessary in the canary script
    browser.quit()

# entry point for the canary
def handler(event, context):
    return custom_selenium_script()
```

Canaries and X-Ray tracing

You can choose to enable active AWS X-Ray tracing on canaries that use the `syn-nodejs-2.0` or later runtime. With tracing enabled, traces are sent for all calls made by the canary that use the browser, the AWS SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the service map in both CloudWatch ServiceLens and in X-Ray, even when they don't send requests to other services or applications that have tracing enabled. For more information about X-Ray tracing, see [Traces](#).

Note

Activating X-Ray tracing on canaries is not yet supported in Asia Pacific (Jakarta).

When a canary appears on a service map, it appears as a new client node type. You can hover on a canary node to see data about latency, requests, and faults. You can also choose the canary node to see more data at the bottom of the page. From this area of the page, you can choose **View in Synthetics** to jump to the CloudWatch Synthetics console for more details about the canary, or choose **View Traces** to see more details about the traces from this canary's runs.

A canary with tracing enabled also has a **Tracing** tab in its details page, with details about traces and segments from the canary's runs.

Enabling tracing increases canary run time by 2.5% to 7%.

A canary with tracing enabled must use a role with the following permissions. If you use the console to create the role when you create the canary, it is given these permissions.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Sid230934",  
            "Effect": "Allow",  
            "Action": [  
                "xray:PutTraceSegments"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Traces generated by canaries incur charges. For more information about X-Ray pricing, see [AWS X-Ray Pricing](#).

Running a canary on a VPC

You can run canaries on endpoints on a VPC and public internal endpoints. To run a canary on a VPC, you must have both the **DNS Resolution** and **DNS hostnames** options enabled on the VPC. For more information, see [Using DNS with Your VPC](#).

When you run a canary on a VPC endpoint, you must provide a way for it to send its metrics to CloudWatch and its artifacts to Amazon S3. If the VPC is already enabled for internet access, there's nothing more for you to do. The canary executes in your VPC, but can access the internet to upload its metrics and artifacts.

If the VPC is not already enabled for internet access, you have two options:

- Enable it for internet access. For more information, see the following section [Giving internet access to your canary on a VPC \(p. 240\)](#).
- If you want to keep your VPC private, you can configure the canary to send its data to CloudWatch and Amazon S3 through private VPC endpoints. If you have not already done so, you must create endpoints for these services on your VPC. For more information, see [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints \(p. 952\)](#) and [Amazon VPC Endpoints for Amazon S3](#).

Giving internet access to your canary on a VPC

Follow these steps to give internet access to your VPC canary.

To give internet access to a canary on a VPC

1. Create a NAT gateway in a public subnet on the VPC. For instructions, see [Create a NAT gateway](#).
2. Add a new route to the route table in the private subnet where the canary is launched. Specify the following:
 - For **Destination**, enter **0.0.0.0/0**
 - For **Target**, choose **NAT Gateway**, and then choose the ID of the NAT gateway that you created.
 - Choose **Save routes**.

For more information about adding the route to the route table, see [Add and remove routes from a route table](#).

Note

Be sure that the routes to your NAT gateway are in an **active** status. If the NAT gateway is deleted and you haven't updated the routes, they're in a blackhole status. For more information, see [Work with NAT gateways](#).

Encrypting canary artifacts

CloudWatch Synthetics stores canary artifacts such as screenshots, HAR files, and reports in your Amazon S3 bucket. By default, these artifacts are encrypted at rest using an AWS managed key. For more information, see [AWS managed keys](#).

You can choose to use a different encryption option. CloudWatch Synthetics supports the following:

- **SSE-S3**– Server-side encryption (SSE) with an Amazon S3-managed key.
- **SSE-KMS**– Server-side encryption (SSE) with an AWS KMS customer managed key.

If you want to use the default encryption option with an AWS managed key, you don't need any additional permissions.

To use SSE-S3 encryption, you specify **SSE_S3** as the encryption mode when you create or update your canary. You do not need any additional permissions to use this encryption mode. For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\)](#).

To use an AWS KMS customer managed key, you specify **SSE-KMS** as the encryption mode when you create or update your canary, and you also provide the Amazon Resource Name (ARN) of your key. You can also use a cross-account KMS key.

To use a customer managed key, you need the following settings:

- The IAM role for your canary must have permission to encrypt your artifacts using your key. If you are using visual monitoring, you must also give it permission to decrypt artifacts.

```
{  
    "Version": "2012-10-17",  
    "Statement": {"Effect": "Allow",  
        "Action": [  
            "kms:GenerateDataKey",  
            "kms:Decrypt"  
        ],  
        "Resource": "Your KMS key ARN"
```

```
}
```

- Instead of adding permissions to your IAM role, you can add your IAM role to your key policy. If you use the same role for multiple canaries, you should consider this approach.

```
{  
    "Sid": "Enable IAM User Permissions",  
    "Effect": "Allow",  
    "Principal": {  
        "AWS": "Your synthetics IAM role ARN"  
    },  
    "Action": [  
        "kms:GenerateDataKey",  
        "kms:Decrypt"  
    ],  
    "Resource": "*"  
}
```

- If you are using a cross-account KMS key, see [Allowing users in other accounts to use a KMS key?](#).

Viewing encrypted canary artifacts when using a customer managed key

To view canary artifacts, update your customer managed key to give AWS KMS the decrypt permission to the user viewing the artifacts. Alternatively, add decrypt permissions to the IAM user or IAM role that is viewing the artifacts.

The default AWS KMS policy enables IAM policies in the account to allow access to the KMS keys. If you are using a cross-account KMS key, see [Why are cross-account users getting Access Denied errors when they try to access Amazon S3 objects encrypted by a custom AWS KMS key?](#)

For more information about troubleshooting access denied issues because of a KMS key, see [Troubleshooting key access](#).

Updating artifact location and encryption when using visual monitoring

To perform visual monitoring, CloudWatch Synthetics compares your screenshots with baseline screenshots acquired in the run selected as the baseline. If you update your artifact location or encryption option, you must do one of the following:

- Ensure that your IAM role has sufficient permission for both the previous Amazon S3 location and the new Amazon S3 location for artifacts. Also ensure that it has permission for both the previous and new encryption methods and KMS keys.
- Create a new baseline by selecting the next canary run as a new baseline. If you use this option, you only need to ensure that your IAM role has sufficient permissions for the new artifact location and encryption option.

We recommend the second option of selecting the next run as the new baseline. This avoids having a dependency on an artifact location or encryption option that you're not using anymore for the canary.

For example, suppose that your canary uses artifact location A and KMS key K for uploading artifacts. If you update your canary to artifact location B and KMS key L, you can ensure that your IAM role has permissions to both of the artifact locations (A and B) and both of the KMS keys (K and L). Alternatively, you can select the next run as the new baseline and ensure that your canary IAM role has permissions to artifact location B and KMS key L.

Viewing canary statistics and details

You can view details about your canaries and see statistics about their runs.

To be able to see all the details about your canary run results, you must be logged on to an account that has sufficient permissions. For more information, see [Required roles and permissions for CloudWatch canaries \(p. 163\)](#).

To view canary statistics and details

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Canaries**.

In the details about the canaries that you have created:

- **Status** visually shows how many of your canaries have passed their most recent runs.
 - In the graph under **Canary runs**, each point represents one minute of your canaries' runs. You can pause on a point to see details.
 - Near the bottom of the page is a table displaying all canaries. A column on the right shows the alarms created for each canary. Only alarms that conform to the naming standard for canary alarms are displayed. This standard is **Synthetics-Alarm-canaryName-index**. Canary alarms that you create in the **Synthetics** section of the CloudWatch console automatically use this naming convention. If you create canary alarms in the **Alarms** section of the CloudWatch console or by using AWS CloudFormation, and you don't use this naming convention, the alarms work but they do not appear in this list.
3. To see more details about a single canary, choose a point in the **Status** graph or choose the name of the canary in the **Canaries** table.

In the details about that canary:

- The **Availability** tab displays information about the recent runs of this canary.

Under **Canary runs**, you can choose one of the lines to see details about that run.

Under the graph, you can choose **Links checked**, **Screenshot**, **HAR file**, or **Logs** to see these types of details. If the canary has active tracing enabled, you can also choose **Traces** to see tracing information from the canary's runs.

The logs for canary runs are stored in S3 buckets and in CloudWatch Logs.

Screenshots show how your customers view your webpages. You can use the HAR files (HTTP Archive files) to view detailed performance data about the webpages. You can analyze the list of web requests and catch performance issues such as time to load for an item. Log files show the record of interactions between the canary run and the webpage and can be used to identify details of errors.

If the canary uses the `syn-nodejs-2.0-beta` runtime or later, you can sort the HAR files by status code, request size, or duration.

If the canary uses the `syn-nodejs-2.0-beta` runtime or later and the canary executes steps in its script, you can choose a **Steps** tab. This tab displays a list of the canary's steps, each step's status, failure reason, URL after step execution, screenshots, and duration of step execution. For API canaries with HTTP steps, you can view steps and corresponding HTTP requests if you are using runtime `syn-nodejs-2.2` or later.

Choose the **HTTP Requests** tab to view the log of each HTTP request made by the canary. You can view request/response headers, response body, status code, error and performance timings (total

duration, TCP connection time, TLS handshake time, first byte time, and content transfer time). All HTTP requests which use the HTTP/HTTPS module under the hood are captured here.

By default in API canaries, the request header, response header, request body, and response body are not included in the report for security reasons. If you choose to include them, the data is stored only in your S3 bucket. For information about how to include this data in the report, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\) \(p. 220\)](#).

Response body content types of text, HTML and JSON are supported. Content types like text/HTML, text/plain, application/JSON and application/x-amz-json-1.0 are supported. Compressed responses are not supported.

- The **Monitoring** tab displays graphs of the CloudWatch metrics published by this canary. For more information about these metrics, see [CloudWatch metrics published by canaries \(p. 244\)](#).

Below the CloudWatch graphics published by the canary are graphs of Lambda metrics related to the canary's Lambda code.

- The **Configuration** tab displays configuration and schedule information about the canary.
- The **Tags** tab displays the tags associated with the canary.

CloudWatch metrics published by canaries

Canaries publish the following metrics to CloudWatch in the `CloudWatchSynthetics` namespace. For more information about viewing CloudWatch metrics, see [Viewing available metrics \(p. 71\)](#).

| Metric | Description |
|----------------|--|
| SuccessPercent | <p>The percentage of the runs of this canary that succeed and find no failures.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Average</p> <p>Units: Percent</p> |
| Duration | <p>The duration in milliseconds of the canary run.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Average</p> <p>Units: Milliseconds</p> |
| 2xx | <p>The number of network requests performed by the canary that returned OK responses, with response codes between 200 and 299.</p> <p>This metric is reported for UI canaries that use runtime version <code>syn-nodejs-2.0</code> or later, and is reported for API canaries that use runtime version <code>syn-nodejs-2.2</code> or later.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p> <p>Units: Count</p> |
| 4xx | <p>The number of network requests performed by the canary that returned Error responses, with response codes between 400 and 499.</p> |

| Metric | Description |
|--|--|
| | <p>This metric is reported for UI canaries that use runtime version <code>syn-nodejs-2.0</code> or later, and is reported for API canaries that use runtime version <code>syn-nodejs-2.2</code> or later.</p> <p>Valid Dimensions: <code>CanaryName</code></p> <p>Valid Statistic: <code>Sum</code></p> <p>Units: <code>Count</code></p> |
| <code>5xx</code> | <p>The number of network requests performed by the canary that returned Fault responses, with response codes between 500 and 599.</p> <p>This metric is reported for UI canaries that use runtime version <code>syn-nodejs-2.0</code> or later, and is reported for API canaries that use runtime version <code>syn-nodejs-2.2</code> or later.</p> <p>Valid Dimensions: <code>CanaryName</code></p> <p>Valid Statistic: <code>Sum</code></p> <p>Units: <code>Count</code></p> |
| <code>Failed</code> | <p>The number of canary runs that failed to execute. These failures are related to the canary itself.</p> <p>Valid Dimensions: <code>CanaryName</code></p> <p>Valid Statistic: <code>Sum</code></p> <p>Units: <code>Count</code></p> |
| <code>Failed requests</code> | <p>The number of HTTP requests executed by the canary on the target website that failed with no response.</p> <p>Valid Dimensions: <code>CanaryName</code></p> <p>Valid Statistic: <code>Sum</code></p> <p>Units: <code>Count</code></p> |
| <code>VisualMonitoringSuccessRate</code> | <p>The percentage of visual comparisons that successfully matched the baseline screenshots during a canary run.</p> <p>Valid Dimensions: <code>CanaryName</code></p> <p>Valid Statistic: <code>Average</code></p> <p>Units: <code>Percent</code></p> |
| <code>VisualMonitoringTotal</code> | <p>The total number of visual comparisons that happened during a canary run.</p> <p>Valid Dimensions: <code>CanaryName</code></p> <p>Units: <code>Count</code></p> |

Note

Canaries that use either the `executeStep()` or `executeHttpStep()` methods from the Synthetics library also publish `SuccessPercent` and `Duration` metrics with the dimensions `CanaryName` and `StepName` for each step.

Editing or deleting a canary

You can edit or delete an existing canary.

Edit canary

When you edit a canary, even if you don't change its schedule, the schedule is reset corresponding to when you edit the canary. For example, if you have a canary that runs every hour, and you edit that canary, the canary will run immediately after the edit is completed and then every hour after that.

To edit or update a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Canaries**.
3. Select the button next to the canary name, and choose **Actions, Edit**.
4. (Optional) If this canary performs visual monitoring of screenshots and you want to set the next run of the canary as the baseline, select **Set next run as new baseline**.
5. (Optional) If this canary performs visual monitoring of screenshots and you want to remove a screenshot from visual monitoring or you want to designate parts of the screenshot to be ignored during visual comparisons, under **Visual Monitoring** choose **Edit Baseline**.

The screenshot appears, and you can do one of the following:

- To remove the screenshot from being used for visual monitoring, select **Remove screenshot from visual test baseline**.
 - To designate parts of the screenshot to be ignored during visual comparisons, click and drag to draw areas of the screen to ignore. Once you have done this for all the areas that you want to ignore during comparisons, choose **Save**.
6. Make any other changes to the canary that you'd like, and choose **Save**.

Delete canary

When you delete a canary, resources used and created by the canary are not automatically deleted. After you delete a canary that you do not intend to use again, you should also delete the following:

- Lambda functions and layers used by this canary. Their prefix is `cwsyn-MyCanaryName`.
- CloudWatch alarms created for this canary. These alarms have a name that starts with `Synthetics-Alarm-MyCanaryName`. For more information about deleting alarms, see [Editing or deleting a CloudWatch alarm \(p. 148\)](#).
- Amazon S3 objects and buckets, such as the canary's results location and artifact location.
- IAM roles created for the canary. These have the name `role/service-role/CloudWatchSyntheticsRole-MyCanaryName`.
- Log groups in CloudWatch Logs created for the canary. These logs groups have the following names: `/aws/lambda/cwsyn-MyCanaryName`.

Before you delete a canary, you might want to view the canary details and make note of this information. That way, you can delete the correct resources after you delete the canary.

To delete a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Canaries**.
3. Select the button next to the canary name, and choose **Actions, Delete**.
4. Enter **Delete** into the box and choose **Delete**.
5. Delete the other resources used by and created for the canary, as listed earlier in this section.

Monitoring canary events with Amazon EventBridge

Amazon EventBridge event rules can notify you when canaries change status or complete runs. EventBridge delivers a near-real-time stream of system events that describe changes in AWS resources. CloudWatch Synthetics sends these events to EventBridge on a *best effort* basis. Best effort delivery means that CloudWatch Synthetics attempts to send all events to EventBridge, but in some rare cases an event might not be delivered. EventBridge processes all received events at least once. Additionally, your event listeners might not receive the events in the order that the events occurred.

Note

Amazon EventBridge is an event bus service that you can use to connect your applications with data from a variety of sources. For more information, see [What is Amazon EventBridge?](#) in the *Amazon EventBridge User Guide*.

CloudWatch Synthetics emits an event when a canary changes state or completes a run. You can create an EventBridge rule that includes an event pattern to match all event types sent from CloudWatch Synthetics, or that matches only specific event types. When a canary triggers a rule, EventBridge invokes the target actions defined in the rule. This allows you to send notifications, capture event information, and take corrective action, in response to a canary state change or the completion of a canary run. For example, you can create rules for the following use cases:

- Investigating when a canary run fails
- Investigating when a canary has gone into the `ERROR` state
- Tracking a canary's life cycle
- Monitoring canary run success or failure as part of a workflow

Example events from CloudWatch Synthetics

This section lists example events from CloudWatch Synthetics. For more information about event format, see [Events and Event Patterns in EventBridge](#).

Canary status change

In this event type, the values of `current-state` and `previous-state` can be the following:

`CREATING | READY | STARTING | RUNNING | UPDATING | STOPPING | STOPPED | ERROR`

```
{  
    "version": "0",  
    "id": "8a99ca10-1e97-2302-2d64-316c5dedfd61",  
    "detail-type": "Synthetics Canary Status Change",  
    "source": "aws.synthetics",  
    "account": "123456789012",  
    "time": "2021-02-09T22:19:43Z",  
}
```

```

"region": "us-east-1",
"resources": [],
"detail": {
    "account-id": "123456789012",
    "canary-id": "EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
    "canary-name": "events-bb-1",
    "current-state": "STOPPED",
    "previous-state": "UPDATING",
    "source-location": "NULL",
    "updated-on": 1612909161.767,
    "changed-config": {
        "executionArn": {
            "previous-value": "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
            "current-value": "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
        },
        "vpcId": {
            "current-value": "NULL"
        },
        "testCodeLayerVersionArn": {
            "previous-
value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
            "current-value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
        }
    },
    "message": "Canary status has changed"
}
}

```

Successful canary run completed

```

{
    "version": "0",
    "id": "989EXAMPLE-f4a5-57a7-1a8f-d9cc768a1375",
    "detail-type": "Synthetics Canary TestRun Successful",
    "source": "aws.synthetics",
    "account": "123456789012",
    "time": "2021-02-09T22:24:01Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "account-id": "123456789012",
        "canary-id": "989EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
        "canary-name": "events-bb-1",
        "canary-run-id": "c6c39152-8f4a-471c-9810-989EXAMPLE",
        "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-28ddbe266797/2021/02/09/22/23-41-200",
        "test-run-status": "PASSED",
        "state-reason": "null",
        "canary-run-timeline": {
            "started": 1612909421,
            "completed": 1612909441
        },
        "message": "Test run result is generated successfully"
    }
}

```

Failed canary run completed

```
{
    "version": "0",
    "id": "2644b18f-3e67-5ebf-cdfd-bf9f91392f41",
    "detail-type": "Synthetics Canary TestRun Failure",
    "source": "aws.synthetics",
    "account": "123456789012",
    "time": "2021-02-09T22:24:27Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "account-id": "123456789012",
        "canary-id": "af3e3a05-dc5a-4f5f-96d1-9989EXAMPLE",
        "canary-name": "events-bb-1",
        "canary-run-id": "0df3823e-7e33-4da1-8194-b04e4d4a2bf6",
        "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-989EXAMPLE/2021/02/09/22/24-21-275",
        "test-run-status": "FAILED",
        "state-reason": "\\" Error: net::ERR_NAME_NOT_RESOLVED \\""
        "canary-run-timeline": {
            "started": 1612909461,
            "completed": 1612909467
        },
        "message": "Test run result is generated successfully"
    }
}
```

It's possible that events might be duplicated or out of order. To determine the order of events, use the `time` property.

Prerequisites for creating EventBridge rules

Before you create an EventBridge rule for CloudWatch Synthetics, you should do the following:

- Familiarize yourself with events, rules, and targets in EventBridge.
- Create and configure the targets invoked by your EventBridge rules. Rules can invoke many types of targets, including:
 - Amazon SNS topics
 - AWS Lambda functions
 - Kinesis streams
 - Amazon SQS queues

For more information, see [What is Amazon EventBridge?](#) and [Getting started with Amazon EventBridge](#) in the *Amazon EventBridge User Guide*.

Create an EventBridge rule (CLI)

The steps in the following example create an EventBridge rule that publishes an Amazon SNS topic when the canary named `my-canary-name` in `us-east-1` completes a run or changes state.

1. Create the rule.

```
aws events put-rule \
--name TestRule \
--region us-east-1 \
--event-pattern "{\"source\": [\"aws.synthetics\"], \"detail\": {\"canary-name\": [
\"my-canary-name\"]}}"
```

- Any properties you omit from the pattern are ignored.
2. Add the topic as a rule target.
 - Replace `topic-arn` with the Amazon Resource Name (ARN) of your Amazon SNS topic.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id=\"1\", Arn=\"topic-arn\""
```

Note

To allow Amazon EventBridge to call your target topic, you must add a resource-based policy to your topic. For more information, see [Amazon SNS permissions](#) in the *Amazon EventBridge User Guide*.

For more information, see [Events and event patterns in EventBridge](#) in the *Amazon EventBridge User Guide*.

Using metric streams

You can use *metric streams* to continually stream CloudWatch metrics to a destination of your choice, with near-real-time delivery and low latency. Supported destinations include AWS destinations such as Amazon Simple Storage Service and several third-party service provider destinations.

There are two main usage scenarios for CloudWatch metric streams:

- **Data lake**— Create a metric stream and direct it to an Amazon Kinesis Data Firehose delivery stream that delivers your CloudWatch metrics to a data lake such as Amazon S3. This enables you to continually update monitoring data, or to combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.
- **Third-party providers**— Use third-party service providers to monitor, troubleshoot, and analyze your applications using the streamed CloudWatch data.

By using filtering options with metric streams, you can stream all your CloudWatch metrics, all your CloudWatch metrics except for namespaces that you choose to omit, or a narrower set of only the namespaces that you select. Each metric stream can include up to 1000 filters that either include or exclude metric namespaces. A single metric stream can have only include or exclude filters, but not both.

After a metric stream is created, if new metrics are created that match the filters in place, the new metrics are automatically included in the stream.

There is no limit on the number of metric streams per account or per Region, and no limit on the number of metric updates being streamed.

Each stream can use either JSON format or OpenTelemetry 0.7.0 format.

Metric streams pricing is based on the number of metric updates. You will also incur charges from Kinesis Data Firehose for the delivery stream used for the metric stream. For more information, see [Amazon CloudWatch Pricing](#).

Topics

- [Setting up a metric stream \(p. 251\)](#)
- [Metric stream operation and maintenance \(p. 254\)](#)
- [Monitoring your metric streams with CloudWatch metrics \(p. 255\)](#)
- [Trust between CloudWatch and Kinesis Data Firehose \(p. 256\)](#)
- [Metric streams output formats \(p. 256\)](#)
- [Troubleshooting \(p. 266\)](#)

Setting up a metric stream

Use the steps in the following sections to set up a CloudWatch metric stream.

After a metric stream is created, the time it takes for metric data to appear at the destination depends on the configured buffering settings on the Kinesis Data Firehose delivery stream. The buffering is expressed in maximum payload size or maximum wait time, whichever is reached first. If these are set to the minimum values (60 seconds, 1MB) the expected latency is within 3 minutes if the selected CloudWatch namespaces have active metric updates.

In a CloudWatch metric stream, data is sent every minute. Data might arrive at the final destination out of order. All metrics in the specified namespaces are sent in the metric stream, except metrics with a timestamp that is more than two hours old.

To create and manage metric streams, you must be logged on to an account that has the **CloudWatchFullAccess** policy and the `iam:PassRole` permission, or an account that has the following list of permissions:

- `iam:PassRole`
- `cloudwatch:PutMetricStream`
- `cloudwatch>DeleteMetricStream`
- `cloudwatch:GetMetricStream`
- `cloudwatch>ListMetricStreams`
- `cloudwatch:StartMetricStreams`
- `cloudwatch:StopMetricStreams`

If you're going to have CloudWatch set up the IAM role needed for metric streams, you must also have the `iam:CreateRole` and `iam:PutRolePolicy` permissions.

Important

A user with the `cloudwatch:PutMetricStream` has access the CloudWatch metric data that is being streamed, even if they don't have the `cloudwatch:GetMetricData` permission.

Topics

- [Setting up a metric stream to an AWS service \(data lake scenario\) \(p. 252\)](#)
- [Setting up a metric stream to a third-party solution \(p. 254\)](#)

Setting up a metric stream to an AWS service (data lake scenario)

You can use the CloudWatch console, the AWS CLI, AWS CloudFormation, or the AWS Cloud Development Kit (CDK) to set up a metric stream.

The Kinesis Data Firehose delivery stream that you use for your metric stream must be in the same account and the same Region where you set up the metric stream. To achieve cross-Region functionality, you can configure the Kinesis Data Firehose delivery stream to stream to a final destination that is in a different account or a different Region.

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream to another AWS service.

When you use the console to set up a metric stream, you have the option of choosing the **Quick S3 Setup**. This method works well if want the final output as JSON or to be queried by Amazon Athena. If you want the final format to be Parquet format or Optimized Row Columnar (ORC), you should create your own Kinesis Data Firehose delivery stream and choose **Select an existing Firehose owned by your account** instead of choosing **Quick S3 Setup**.

To set up a metric stream to an AWS service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Streams, Create stream**.

3. Choose the CloudWatch metric namespaces to include in the metric stream.
 - To include all or most of your metric namespaces in the metric stream, choose **All metrics**. Then, if you want to exclude some metric namespaces from the stream, choose **Exclude metric namespaces** and select the namespaces to exclude.
 - To include only a few metric namespaces in the metric stream, choose **Selected namespaces** and then select the namespaces to include.
- Note**

Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.
4. Choose one of the following:
 - For a quick setup, choose **Quick S3 setup**. CloudWatch will create all necessary resources including the Kinesis Data Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format later in this procedure.
 - To use a Kinesis Data Firehose delivery stream that already exists, choose **Select an existing Firehose owned by your account**. The Kinesis Data Firehose delivery stream must be in the same account. The default format for this option is OpenTelemetry, but you can change the format later in this procedure.
- Then select the Kinesis Data Firehose delivery stream to use under **Select your Kinesis Data Firehose delivery stream**.
5. (Optional) If you're using **Quick S3 Setup**, you can choose **Select existing resources** to use an existing S3 bucket or existing IAM roles instead of having CloudWatch create new ones for you.
6. (Optional) If you're using **Select an existing Firehose owned by your account**, you can choose **Select existing service role** to use an existing IAM role instead of having CloudWatch create a new one for you.
7. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON and OpenTelemetry 0.7.0.
8. (Optional) Customize the name of the new metric stream under **Metric stream name**.
9. Choose **Create metric stream**.

AWS CLI or AWS API

Use the following steps to create a CloudWatch metric stream to another AWS service.

To use the AWS CLI or AWS API to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see [Creating a bucket](#).
2. Create the Kinesis Data Firehose delivery stream. For more information, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#).
3. Create an IAM role that enables CloudWatch to write to the Kinesis Data Firehose delivery stream. For more information about the contents of this role, see [Trust between CloudWatch and Kinesis Data Firehose \(p. 256\)](#).
4. Use the `aws cloudwatch put-metric-stream` CLI command or the `PutMetricStream` API to create the CloudWatch metric stream.

AWS CloudFormation

You can use AWS CloudFormation to set up a metric stream. For more information, see [AWS::CloudWatch::MetricStream](#).

To use AWS CloudFormation to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see [Creating a bucket](#).
2. Create the Kinesis Data Firehose delivery stream. For more information, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#).
3. Create an IAM role that enables CloudWatch to write to the Kinesis Data Firehose delivery stream. For more information about the contents of this role, see [Trust between CloudWatch and Kinesis Data Firehose \(p. 256\)](#).
4. Create the stream in AWS CloudFormation. For more information, see [AWS::CloudWatch::MetricStream](#).

AWS Cloud Development Kit (CDK)

You can use AWS Cloud Development Kit (CDK) to set up a metric stream.

To use the AWS CDK to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see [Creating a bucket](#).
2. Create the Kinesis Data Firehose delivery stream. For more information, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#).
3. Create an IAM role that enables CloudWatch to write to the Kinesis Data Firehose delivery stream. For more information about the contents of this role, see [Trust between CloudWatch and Kinesis Data Firehose \(p. 256\)](#).
4. Create the metric stream. The metric stream resource is available in AWS CDK as a Level 1 (L1) Construct named `CfnMetricStream`. For more information, see [Using L1 constructs](#).

Setting up a metric stream to a third-party solution

To set up a CloudWatch metric stream to stream metrics to a third-party solution, follow the instructions provided by that third-party provider.

Metric stream operation and maintenance

Metric streams are always in one of two states, **Running** or **Stopped**.

- **Running**— The metric stream is running correctly. There might not be any metric data streamed to the destination because of the filters on the stream.
- **Stopped**— The metric stream has been explicitly stopped by someone, and not because of an error. It might be useful to stop your stream to temporarily pause the streaming of data without deleting the stream.

If you stop and restart a metric stream, the metric data that was published to CloudWatch while the metric stream was stopped is not backfilled to the metric stream.

If you change the output format of a metric stream, in certain cases you might see a small amount of metric data written to the destination in both the old format and the new format. To avoid this situation, you can create a new Kinesis Data Firehose delivery stream with the same configuration as your current one, then change to the new Kinesis Data Firehose delivery stream and change the output format at

the same time. This way, the Kinesis records with different output format are stored on Amazon S3 in separate objects. Later, you can direct the traffic back to the original Kinesis Data Firehose delivery stream and delete the second delivery stream.

To view, edit, stop, and start your metric streams

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Streams**.

The list of streams appears, and the **Status** column displays whether each stream is running or stopped.

3. To stop or start a metric stream, select the stream and choose **Stop** or **Start**.
4. To see the details about a metric stream, select the stream and choose **View details**.
5. To change the stream's output format, filters, destination Kinesis Data Firehose stream, or roles, choose **Edit** and make the changes that you want.

If you change the filters, there might be some gaps in the metric data during the transition.

Monitoring your metric streams with CloudWatch metrics

Metric streams emit CloudWatch metrics about their health and operation in the AWS/Cloudwatch/MetricStreams namespace. The following metrics are emitted. Both metrics are emitted with a MetricStreamName dimension and with no dimension. You can use the metrics with no dimensions to see aggregated metrics for all of your metric streams. You can use the metrics with the MetricStreamName dimension to see the metrics about only that metric stream.

For both of these metrics, values are emitted only for metric streams that are in the **Running** state.

| Metric | Description |
|-------------------------------|---|
| <code>MetricUpdate</code> | <p>The number of metric updates sent to the metric stream. If no metric updates are streamed during a time period, a value of 0 is emitted for this metric.</p> <p>If you stop the metric stream, this metric stops being emitted until the metric stream is started again.</p> <p>Valid Statistic: Sum</p> <p>Units: None</p> |
| <code>PublishErrorRate</code> | <p>The number of unrecoverable errors that occur when putting data into the Kinesis Data Firehose delivery stream. If no errors occur during a time period, a value of 0 is emitted for this metric.</p> <p>If you stop the metric stream, this metric stops being emitted until the metric stream is started again.</p> <p>Valid Statistic: Average to see the rate of metric updates unable to be written. This value will be between 0.0 and 1.0.</p> <p>Units: None</p> |

Trust between CloudWatch and Kinesis Data Firehose

The Kinesis Data Firehose delivery stream must trust CloudWatch through an IAM role that has write permissions to Kinesis Data Firehose. These permissions can be limited to the single Kinesis Data Firehose delivery stream that the CloudWatch metric stream uses. The IAM role must trust the `streams.metrics.cloudwatch.amazonaws.com` service principal.

If you use the CloudWatch console to create a metric stream, you can have CloudWatch create the role with the correct permissions. If you use another method to create a metric stream, or you want to create the IAM role itself, it must contain the following permissions policy and trust policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "firehose:PutRecord",  
                "firehose:PutRecordBatch"  
            ],  
            "Effect": "Allow",  
            "Resource": "arn:aws:firehose:region:account-id:deliverystream/*"  
        }  
    ]  
}
```

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "streams.metrics.cloudwatch.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Metric data is streamed by CloudWatch to the destination Kinesis Data Firehose delivery stream on behalf of the source that owns the metric stream resource.

Metric streams output formats

The data in a CloudWatch metric stream can be in the JSON format or the OpenTelemetry format. Currently, the version of OpenTelemetry format supported is 0.7.0.

JSON format

In a CloudWatch metric stream that uses the JSON format, each Kinesis Data Firehose record contains multiple JSON objects separated by a newline character (\n). Each object includes a single data point of a single metric.

The JSON format that is used is fully compatible with AWS Glue and with Amazon Athena. If you have a Kinesis Data Firehose delivery stream and an AWS Glue table formatted correctly, the format can be

automatically transformed into Parquet format or Optimized Row Columnar (ORC) format before being stored in S3. For more information about transforming the format, see [Converting Your Input Record Format in Kinesis Data Firehose](#). For more information about the correct format for AWS Glue, see [Which AWS Glue schema should I use for JSON output format? \(p. 257\)](#).

In the JSON format, the valid values for `unit` are the same as for the value of `unit` in the `MetricDatum` API structure. For more information, see [MetricDatum](#). The value for the `timestamp` field is in epoch milliseconds, such as 1616004674229.

The following is an example of the format. In this example, the JSON is formatted for easy reading, but in practice the whole format is on a single line.

```
{  
    "metric_stream_name": "MyMetricStream",  
    "account_id": "1234567890",  
    "region": "us-east-1",  
    "namespace": "AWS/EC2",  
    "metric_name": "DiskWriteOps",  
    "dimensions": {  
        "InstanceId": "i-123456789012"  
    },  
    "timestamp": 1611929698000,  
    "value": {  
        "count": 3.0,  
        "sum": 20.0,  
        "max": 18.0,  
        "min": 0.0  
    },  
    "unit": "Seconds"  
}\n
```

Which AWS Glue schema should I use for JSON output format?

The following is an example of a JSON representation of the `StorageDescriptor` for an AWS Glue table, which would then be used by Kinesis Data Firehose. For more information about `StorageDescriptor`, see [StorageDescriptor](#).

```
{  
    "Columns": [  
        {  
            "Name": "metric_stream_name",  
            "Type": "string"  
        },  
        {  
            "Name": "account_id",  
            "Type": "string"  
        },  
        {  
            "Name": "region",  
            "Type": "string"  
        },  
        {  
            "Name": "namespace",  
            "Type": "string"  
        },  
        {  
            "Name": "metric_name",  
            "Type": "string"  
        },  
        {  
            "Name": "timestamp",  
            "Type": "timestamp"  
        }  
    ]  
}
```

```

    },
    {
      "Name": "dimensions",
      "Type": "map<string,string>"
    },
    {
      "Name": "value",
      "Type": "struct<min:double,max:double,count:double,sum:double>"
    },
    {
      "Name": "unit",
      "Type": "string"
    }
  ],
  "Location": "s3://my-s3-bucket/",
  "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
  "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
  "SerdeInfo": {
    "SerializationLibrary": "org.apache.hive.hcatalog.data.JsonSerDe"
  },
  "Parameters": {
    "classification": "json"
  }
}

```

The preceding example is for data written on Amazon S3 in JSON format. Replace the values in the following fields with the indicated values to store the data in Parquet format or Optimized Row Columnar (ORC) format.

- **Parquet:**

- inputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
- outputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
- SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
- parameters.classification: parquet

- **ORC:**

- inputFormat: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
- outputFormat: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
- SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.orc.OrcSerde
- parameters.classification: orc

OpenTelemetry 0.7.0 format

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see [OpenTelemetry](#).

For information about the full OpenTelemetry 0.7.0 specification, see [v0.7.0 release](#).

Note

While metric streams is in general availability, the OpenTelemetry format 0.7.0 is not yet generally available. Metric streams will continue to offer OpenTelemetry format 0.7.0 even when the OpenTelemetry format version 1.0 becomes available.

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```

resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "2345678901"
      }
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  instrumentation_library_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "1"
      double_summary {
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
          labels {
            key: "TableName"
            value: "MyTable"
          }
          start_time_unix_nano: 16049484000000000000
          time_unix_nano: 16049484600000000000
          count: 1
          sum: 1.0
          quantile_values {
            quantile: 0.0
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
        }
      }
      data_points {
        labels {

```

```
        key: "Namespace"
        value: "AWS/DynamoDB"
    }
    labels {
        key: "MetricName"
        value: "ConsumedReadCapacityUnits"
    }
    labels {
        key: "TableName"
        value: "MyTable"
    }
    start_time_unix_nano: 16049484600000000000
    time_unix_nano: 16049485200000000000
    count: 2
    sum: 5.0
    quantile_values {
        quantile: 0.0
        value: 2.0
    }
    quantile_values {
        quantile: 1.0
        value: 3.0
    }
}
}
}
}
```

Top-level object to serialize OpenTelemetry metric data

`ExportMetricsServiceRequest` is the top-level wrapper to serialize an OpenTelemetry exporter payload. It contains one or more `ResourceMetrics`.

```
message ExportMetricsServiceRequest {
    // An array of ResourceMetrics.
    // For data coming from a single resource this array will typically contain one
    // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
    // data from multiple origins typically batch the data before forwarding further and
    // in that case this array will contain multiple elements.
    repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

`ResourceMetrics` is the top-level object to represent `MetricData` objects.

```
// A collection of InstrumentationLibraryMetrics from a Resource.
message ResourceMetrics {
    // The resource for the metrics in this message.
    // If this field is not set then no resource info is known.
    opentelemetry.proto.resource.v1.Resource resource = 1;

    // A list of metrics that originate from a resource.
    repeated InstrumentationLibraryMetrics instrumentation_library_metrics = 2;
}
```

The Resource object

A Resource object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by AWS, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The `Resource` object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
    // Set of labels that describe the resource.
    repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

    // dropped_attributes_count is the number of dropped attributes. If the value is 0,
    // no attributes were dropped.
    uint32 dropped_attributes_count = 2;
}
```

The InstrumentationLibraryMetrics object

The instrumentation_library field will not be filled. We will fill only the metrics field that we are exporting.

```
// A collection of Metrics produced by an InstrumentationLibrary.
message InstrumentationLibraryMetrics {
    // The instrumentation library information for the metrics in this message.
    // If this field is not set then no library info is known.
    opentelemetry.proto.common.v1.InstrumentationLibrary instrumentation_library = 1;
    // A list of metrics that originate from an instrumentation library.
    repeated Metric metrics = 2;
}
```

The Metric object

The metric object contains a `DoubleSummary` data field that contains a list of `DoubleSummaryDataPoint`.

```
message Metric {
    // name of the metric, including its DNS name prefix. It must be unique.
    string name = 1;

    // description of the metric, which can be used in documentation.
    string description = 2;

    // unit in which the metric value is reported. Follows the format
    // described by http://unitsofmeasure.org/ucum.html.
    string unit = 3;

    oneof data {
        IntGauge int_gauge = 4;
        DoubleGauge double_gauge = 5;
        IntSum int_sum = 6;
        DoubleSum double_sum = 7;
        IntHistogram int_histogram = 8;
        DoubleHistogram double_histogram = 9;
        DoubleSummary double_summary = 11;
    }
}

message DoubleSummary {
    repeated DoubleSummaryDataPoint data_points = 1;
}
```

The MetricDescriptor object

The MetricDescriptor object contains metadata. For more information, see [metrics.proto](#) on GitHub.

For metric streams, the MetricDescriptor has the following contents:

- name will be `amazonaws.com/metric_namespace/metric_name`
- description will be blank.
- unit will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see [Translations with OpenTelemetry 0.7.0 format \(p. 263\)](#) and [The Unified Code For Units of Measure](#).
- type will be SUMMARY.

The DoubleSummaryDataPoint object

The DoubleSummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// DoubleSummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message DoubleSummaryDataPoint {
    // The set of labels that uniquely identify this timeseries.
    repeated opentelemetry.proto.common.v1.StringKeyValue labels = 1;

    // start_time_unix_nano is the last time when the aggregation value was reset
    // to "zero". For some metric types this is ignored, see data types for more
    // details.
    //
    // The aggregation value is over the time interval (start_time_unix_nano,
    // time_unix_nano].
    //
    // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
    // 1970.
    //
    // Value of 0 indicates that the timestamp is unspecified. In that case the
    // timestamp may be decided by the backend.
    fixed64 start_time_unix_nano = 2;

    // time_unix_nano is the moment when this aggregation value was reported.
    //
    // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
    // 1970.
    fixed64 time_unix_nano = 3;

    // count is the number of values in the population. Must be non-negative.
    fixed64 count = 4;

    // sum of the values in the population. If count is zero then this field
    // must be zero.
    double sum = 5;

    // Represents the value at a given quantile of a distribution.
    //
    // To record Min and Max values following conventions are used:
    // - The 1.0 quantile is equivalent to the maximum value observed.
    // - The 0.0 quantile is equivalent to the minimum value observed.
    message ValueAtQuantile {
        // The quantile of a distribution. Must be in the interval
        // [0.0, 1.0].
        double quantile = 1;

        // The value at the given quantile of a distribution.
    }
}
```

```

        double value = 2;
    }

    // (Optional) list of values at different quantiles of the distribution calculated
    // from the current snapshot. The quantiles must be strictly increasing.
    repeated ValueAtQuantile quantile_values = 6;
}

```

For more information, see [Translations with OpenTelemetry 0.7.0 format \(p. 263\)](#).

Translations with OpenTelemetry 0.7.0 format

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One pair contains the namespace of the metric
- One pair contains the name of the metric
- For each dimension, CloudWatch stores the following pair: `metricDatum.Dimensions[i].Name`, `metricDatum.Dimensions[i].Value`

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`
- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
 - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`
 - `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see [The Unified Code For Units of Measure](#).

| CloudWatch | OpenTelemetry |
|-------------------|---------------|
| Second | s |
| Second or Seconds | s |
| Microsecond | us |
| Milliseconds | ms |
| Bytes | By |
| Kilobytes | kBy |
| Megabytes | MBy |

| CloudWatch | OpenTelemetry |
|------------|---------------|
| Gigabytes | GBy |
| Terabytes | TBy |
| Bits | bit |
| Kilobits | kbit |
| Megabits | MBit |
| Gigabits | GBit |
| Terabits | Tbit |
| Percent | % |
| Count | {Count} |
| None | 1 |

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 0.7.0 messages

This section provides information to help you get started with parsing OpenTelemetry 0.7.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 0.7.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
 - To use Java, add the following Maven dependency to your Java project: [OpenTelemetry Java >> 0.14.1](#).
 - To use any other language, follow these steps:
 - a. Make sure that your language is supported by checking the list at [Generating Your Classes](#).
 - b. Install the Protobuf compiler by following the steps at [Download Protocol Buffers](#).
 - c. Download the OpenTelemetry 0.7.0 ProtoBuf definitions at [v0.7.0 release](#).
 - d. Confirm that you are in the root folder of the downloaded OpenTelemetry 0.7.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see [Generating Your Classes](#).

The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```

package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
    IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest` records, each of them starting with a header with an UnsignedVarInt32 indicating the record length in bytes:
        -----
        |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
        -----
        */
        while ((request = ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) !=
null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}

```

Javascript

This example assumes that the root folder with the bindings generated is ./

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array`.`number` 8-bit integers

```

const pb = require('google-protobuf')
const pbMetrics =
    require('../opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest` records, each of them starting with a header with an UnsignedVarInt32 indicating the record length in bytes:
        -----
        |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
        -----
        */
        const reader = new pb.BinaryReader(data)

```

```
const messageLength = reader.decoder_.readUnsignedVarint32()
const messageFrom = reader.decoder_.cursor_
const messageTo = messageFrom + messageLength

// Extract the current `ExportMetricsServiceRequest` message to parse
const message = data.subarray(messageFrom, messageTo)

// Parse the current message using the ProtoBuf library
const parsed =
    pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

// Do whatever we want with the parsed message
result.push(parsed.toObject())

// Shrink the remaining buffer, removing the already parsed data
data = data.subarray(messageTo)
}

return result
}
```

Python

You must read the var-int delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see [Download Protocol Buffers](#).

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest` Protocol Buffers messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

Troubleshooting

If you're not seeing metric data at your final destination, check the following:

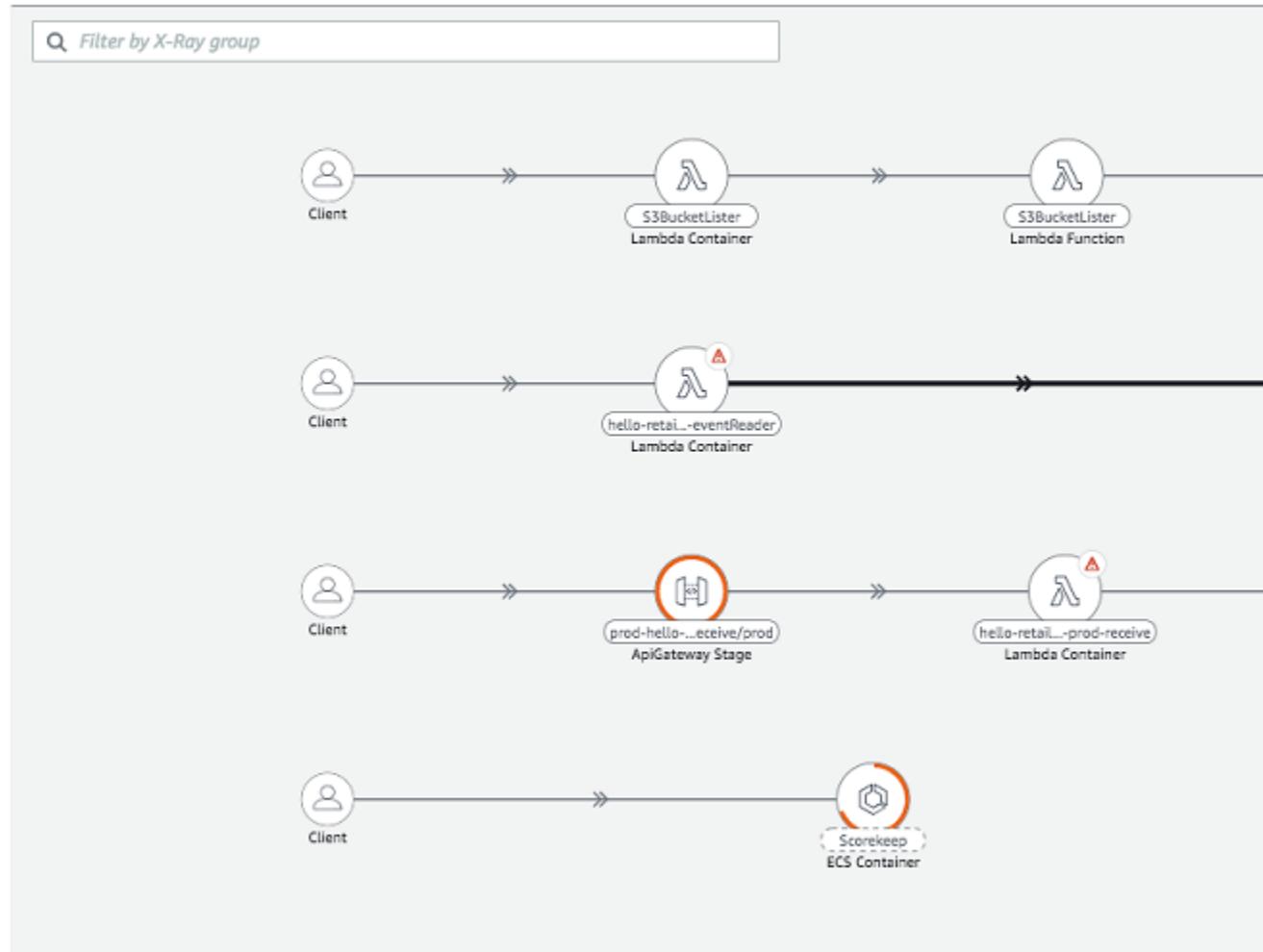
- Check that the metric stream is in the running state. For steps on how to use the CloudWatch console to do this, see [Metric stream operation and maintenance \(p. 254\)](#).
- Metrics published more than two hours in the past are not streamed. To determine whether a particular metric will be streamed, graph the metric in the CloudWatch console and check how old the last visible datapoint is. If it is more than two hours in the past, then it won't be picked up by metric streams.
- Check the metrics emitted by the metric stream. In the CloudWatch console, under **Metrics**, look at the **AWS/CloudWatch/MetricStreams** namespace for the **MetricUpdate** and **PublishErrorRate** metrics.
- If the **PublishErrorRate** metric is high, confirm that the destination that is used by the Kinesis Data Firehose delivery stream exists and that the IAM role specified in the metric stream's configuration grants the CloudWatch service principal permissions to write to it. For more information, see [Trust between CloudWatch and Kinesis Data Firehose \(p. 256\)](#).
- Check that the Kinesis Data Firehose delivery stream has permission to write to the final destination.
- In the Kinesis Data Firehose console, view the Kinesis Data Firehose delivery stream that is used for the metric stream and check the **Monitoring** tab to see whether the Kinesis Data Firehose delivery stream is receiving data.
- Confirm that you have configured your Kinesis Data Firehose delivery stream with the correct details.
- Check any available logs or metrics for the final destination that the Kinesis Data Firehose delivery stream writes to.
- To get more detailed information, enable CloudWatch Logs error logging on the Kinesis Data Firehose delivery stream. For more information, see [Monitoring Kinesis Data Firehose Using CloudWatch Logs](#).

Using ServiceLens to monitor the health of your applications

CloudWatch ServiceLens enhances the observability of your services and applications by enabling you to integrate traces, metrics, logs, alarms, and other resource health information into one place. ServiceLens integrates CloudWatch with AWS X-Ray to provide an end-to-end view of your application to help you more efficiently pinpoint performance bottlenecks and identify impacted users. A service map displays your service endpoints and resources as “nodes” and highlights the traffic, latency, and errors for each node and its connections. You can choose a node to see detailed insights about the correlated metrics, logs, and traces associated with that part of the service. This enables you to investigate problems and their effect on the application.

- [CloudWatch](#) > Service Map

Service map



To fully take advantage of ServiceLens and correlated metrics, logs, and traces, you must update the X-Ray SDK and the instrumentation of your application. ServiceLens supports logs correlation for Lambda

functions, API Gateway, Java-based applications running on Amazon EC2, and Java-based applications running on Amazon EKS or Kubernetes with Container Insights deployed.

ServiceLens integrates with Amazon CloudWatch Synthetics, a fully-managed service that enables you to create canaries to monitor your endpoints and APIs from the outside-in. Canaries that you create appear on the ServiceLens service map. For more information, see [Using synthetic monitoring \(p. 162\)](#).

ServiceLens is available in every Region where X-Ray is available.

Topics

- [Deploying ServiceLens \(p. 269\)](#)
- [Using the service map in ServiceLens \(p. 277\)](#)
- [Using the traces view in ServiceLens \(p. 279\)](#)
- [Using the resource health view in ServiceLens \(p. 279\)](#)
- [ServiceLens troubleshooting \(p. 281\)](#)

Deploying ServiceLens

Deploying ServiceLens requires two steps:

- Deploy AWS X-Ray so that you can view the service map.
- Deploy the CloudWatch agent and the X-Ray daemon to enable the service map integration with CloudWatch metrics and CloudWatch Logs.

Topics

- [Deploying AWS X-Ray \(p. 269\)](#)
- [Deploying the CloudWatch agent and the X-Ray daemon \(p. 271\)](#)

Deploying AWS X-Ray

You can use any AWS X-Ray SDK to enable X-Ray. However, the correlation of logs and metrics with your traces is supported only if you use the Java SDK.

To deploy X-Ray, follow the standard X-Ray setup. For more information, see the following:

- [AWS X-Ray SDK for Java](#) (supports logs correlations)
- [The X-Ray SDK for Node.js](#)
- [AWS X-Ray SDK for .NET](#)
- [AWS X-Ray SDK for Go](#)
- [AWS X-Ray SDK for Python](#)
- [AWS X-Ray SDK for Ruby](#)

After completing the X-Ray setup, follow the steps in the following sections to integrate X-Ray with CloudWatch Logs and enable segment metrics.

Topics

- [Integrating with CloudWatch Logs \(p. 270\)](#)
- [Enabling segment metrics from X-Ray \(p. 270\)](#)

Integrating with CloudWatch Logs

To enable integration with CloudWatch Logs, there are two steps:

- Enable trace to logs correlation. This is supported only using the SDK for Java.
- Configure trace ID injection.

Enabling trace to logs correlation

The SDK for Java supports both a set of standard application logging frameworks and CloudWatch Logs native support. Before completing the following steps, you must have completed a standard setup of the AWS X-Ray SDK for Java.

The supported runtimes are Amazon EC2, Amazon ECS with CloudWatch Container Insights enabled, and Lambda.

- To enable trace to logs correlation on Amazon EC2, enable the X-Ray EC2 Plugin. For more information, see [Service Plugins](#)
- To enable trace to logs correlation on Amazon EKS, first enable Container Insights if you have not already done so. For more information, see [Using Container Insights \(p. 305\)](#).

Then, enable the X-Ray SDK EKS Plugin. For more information, see [Service Plugins](#).

- To enable trace to logs correlation on Lambda, you must enable X-Ray on Lambda. For more information, see [AWS Lambda and AWS X-Ray](#).

Enabling trace ID injection

For information about how to enable trace ID injection, see [Logging](#).

Enabling segment metrics from X-Ray

The AWS X-Ray SDK for Java can emit several metrics about segments into CloudWatch to give an unsampled view of latency, throttle, error, and fault rates. It uses the CloudWatch agent to emit these metrics to minimize the impact on application performance. For more information about segments, see [Segments](#).

If you enable segment metrics, a log group called **XRayApplicationMetrics** is created, and the metrics **ErrorRate**, **FaultRate**, **ThrottleRate**, and **Latency**, are published into a custom CloudWatch metric namespace called **Observability**.

Segment metrics are not currently supported in Lambda.

To enable the AWS X-Ray SDK for Java to publish segment metrics, use the following example.

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder.standard().withSegmentListener(new MetricsSegmentListener());
```

If you are using ServiceLens with Amazon EKS and Container Insights, add the **AWS_XRAY_METRICS_DAEMON_ADDRESS** environment variable to the **HOST_IP** as shown in the following example.

```
env:  
- name: HOST_IP  
  valueFrom:  
    fieldRef:  
      apiVersion: v1
```

```
    fieldPath: status.hostIP
- name: AWS_XRAY_METRICS_DAEMON_ADDRESS
  value: ${HOST_IP}:25888
```

For more information, see [Enable X-Ray CloudWatch Metrics](#).

Deploying the CloudWatch agent and the X-Ray daemon

This section explains how to deploy the CloudWatch agent and the X-Ray daemon. You can deploy the agent and the daemon in the following environments:

- Amazon ECS or Fargate
- Amazon EKS or Kubernetes hosted on Amazon EC2
- Amazon EC2

The deployment steps for each of these environments are explained in the following sections.

Topics

- [Deploying the CloudWatch agent and the X-Ray daemon on Amazon ECS \(p. 271\)](#)
- [Deploying the CloudWatch agent and the X-Ray daemon on Amazon EKS or Kubernetes \(p. 275\)](#)
- [Deploying the CloudWatch agent and the X-Ray daemon on Amazon EC2 \(p. 277\)](#)

Deploying the CloudWatch agent and the X-Ray daemon on Amazon ECS

On Amazon ECS, you deploy the CloudWatch agent as a sidecar to your application container to collect metrics. You can configure the CloudWatch Agent through SSM parameter store.

Creating IAM roles

You must create two IAM roles. If you already have created these roles, you may need to add permissions to them.

- **ECS task role**— Containers use this role to run. The permissions should be whatever your applications need, plus **CloudWatchAgentServerPolicy** and **AWSXRayDaemonWriteAccess**.
- **ECS task execution role**— Amazon ECS uses this role to launch and execute your containers. If you have already created this role, attach the **AmazonSSMReadOnlyAccess**, **AmazonECSTaskExecutionRolePolicy**, and **CloudWatchAgentServerPolicy** policies to it.

If you need to store more sensitive data for Amazon ECS to use, see [Specifying Sensitive Data](#) for more information.

For more information about creating IAM roles, see [Creating IAM Roles](#).

Store the agent configuration in SSM Parameter Store

You need to make sure your agent configuration file has the following section, and then upload it to the SSM parameter store.

```
{
  "logs": {
```

```
        "metrics_collected": {
            "emf": {}
        }
    }
```

To upload the agent configuration to the SSM parameter store

1. Put the agent configuration content into a local file /tmp/ecs-cwagent.json
2. Enter the following command. Replace `region` with the Region of your cluster.

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "region"
```

Create a task definition and launch the task

The steps for this task depend on whether you want to use the EC2 launch type or the Fargate launch type.

EC2 launch type

First, create the task definition. In this example, the container "demo-app" sends X-Ray SDK metrics to the CloudWatch agent and sends trace information to the X-Ray daemon.

Copy the following task definition to a local JSON file such as /tmp/ecs-cwagent-ec2.json. Replace the following placeholders:

- Replace `{ecs-task-role}` with the ARN of your ECS task role.
- Replace `{ecs-task-execution-role}` with the ARN of your ECS task execution role.
- Replace `{demo-app-image}` with your application image that has X-Ray SDK integration enabled. Change the name from demo-app to your own application name.
- Replace `{region}` with the name of the AWS Region where you want to send the logs for containers. For example, us-west-2.

```
{
    "family": "ecs-cwagent-ec2",
    "taskRoleArn": "{ecs-task-role}",
    "executionRoleArn": "{ecs-task-execution-role}",
    "networkMode": "bridge",
    "containerDefinitions": [
        {
            "name": "demo-app",
            "image": "{demo-app-image}",
            "links": [
                "cloudwatch-agent",
                "xray-daemon"
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "True",
                    "awslogs-group": "/ecs/ecs-cwagent-ec2",
                    "awslogs-region": "{region}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ]
}
```

```

        },
        {
            "name": "xray-daemon",
            "image": "public.ecr.aws/xray/aws-xray-daemon:latest",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "True",
                    "awslogs-group": "/ecs/ecs-cwagent-ec2",
                    "awslogs-region": "{{region}}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        },
        {
            "name": "cloudwatch-agent",
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "secrets": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "ecs-cwagent"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "True",
                    "awslogs-group": "/ecs/ecs-cwagent-ec2",
                    "awslogs-region": "{{region}}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ],
    "requiresCompatibilities": [
        "EC2"
    ],
    "cpu": "256",
    "memory": "256"
}

```

Enter the following command to create the task definition. Replace `{{region}}` with the Region of your cluster.

```
aws ecs register-task-definition \
--cli-input-json file:///tmp/ecs-cwagent-ec2.json \
--region {{region}}
```

Enter the following command to launch the task. Replace `{{cluster-name}}` and `{{region}}` with the name and Region of your cluster.

```
aws ecs run-task \
--cluster {{cluster-name}} \
--task-definition ecs-cwagent-ec2 \
--region {{region}} \
--launch-type EC2
```

Fargate launch type

First, create the task definition. In this example, the container “demo-app” sends X-Ray SDK metrics to the CloudWatch agent and sends trace information to the X-Ray daemon.

Copy the following task definition to a local JSON file such as `/tmp/ecs-cwagent-ec2.json`. Replace the following placeholders:

- Replace `{{ecs-task-role}}` with the Amazon Resource Name (ARN) of your ECS task role.
- Replace `{{ecs-task-execution-role}}` with the ARN of your ECS task execution role.
- Replace `{{demo-app-image}}` with your application image that has X-Ray SDK integration enabled. Change the name from demo-app to your own application name.
- Replace `{{region}}` with the name of the AWS Region where you want to send the logs for containers. For example, `us-west-2`.

```
{
    "family": "ecs-cwagent-fargate",
    "taskRoleArn": "{{ecs-task-role}}",
    "executionRoleArn": "{{ecs-task-execution-role}}",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "demo-app",
            "image": "{{demo-app-image}}",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "True",
                    "awslogs-group": "/ecs/ecs-cwagent-fargate",
                    "awslogs-region": "{{region}}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        },
        {
            "name": "xray-daemon",
            "image": "public.ecr.aws/xray/aws-xray-daemon:latest",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "True",
                    "awslogs-group": "/ecs/ecs-cwagent-fargate",
                    "awslogs-region": "{{region}}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        },
        {
            "name": "cloudwatch-agent",
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "secrets": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "ecs-cwagent"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "True",
                    "awslogs-group": "/ecs/ecs-cwagent-fargate",
                    "awslogs-region": "{{region}}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ]
}
```

```
    "requiresCompatibilities": [  
        "FARGATE"  
    ],  
    "cpu": "512",  
    "memory": "1024"  
}
```

Enter the following command to create the task definition. Replace `{{region}}` with the Region of your cluster.

```
aws ecs register-task-definition \  
    --cli-input-json file:///tmp/ecs-cwagent-fargate.json \  
    --region {{region}}
```

If you already have a Fargate cluster set up, you can use the task definition you just created to launch the task. If you do not yet have any Fargate clusters, see [Configure the Service](#) for more information about the rest of the steps to set up Fargate.

Deploying the CloudWatch agent and the X-Ray daemon on Amazon EKS or Kubernetes

These topics explain how to install the X-Ray daemon and the CloudWatch agent on Amazon EKS or Kubernetes.

Deploying the X-Ray daemon on Amazon EKS or Kubernetes

To install the CloudWatch agent and the X-Ray daemon on Amazon EKS or Kubernetes, you can use a quick setup.

To install the CloudWatch agent and the X-Ray daemon on Amazon EKS or Kubernetes

1. Ensure that the IAM role that is attached to the EC2 instance, or the Kubernetes worker node, has the **CloudWatchAgentServerPolicy** and **AWSXRayDaemonWriteAccess** policies attached.
2. Enter the following command:

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-  
insights/master/k8s-deployment-manifest-templates/deployment-mode/daemonset/cwagent-  
fluentd-xray/cwagent-fluentd-xray-quickstart.yaml | sed "s/{{cluster_name}}/cluster-  
name/;s/{{region_name}}/region/" | kubectl apply -f -
```

What the Quick Start does

This section describes the quick setup of the CloudWatch agent and the X-Ray daemon.

- The quick setup specifies inbound ports and protocols. Outbound connections do not have to be explicitly opened.
- The quick setup installs the X-Ray daemon via the `kubectl apply -f` command with the following file content.

```
apiVersion: apps/v1  
kind: DaemonSet  
metadata:  
  name: xray-daemon  
  namespace: amazon-cloudwatch  
spec:  
  selector:
```

```

matchLabels:
  name: xray-daemon
template:
  metadata:
    labels:
      name: xray-daemon
spec:
  containers:
    - name: xray-daemon
      image: amazon/aws-xray-daemon:latest
      imagePullPolicy: Always
      ports:
        - containerPort: 2000
          hostPort: 2000
          protocol: UDP
      resources:
        limits:
          cpu: 100m
          memory: 256Mi
        requests:
          cpu: 50m
          memory: 50Mi
  terminationGracePeriodSeconds: 60

```

- The quick setup updates the CloudWatch agent using the Docker image version/label 1.231221.0 or later, or the latest version. You can find the image at <https://hub.docker.com/r/amazon/cloudwatch-agent>.
- To enable the X-Ray SDK to read cluster name and Region information, the quick setup updates the CloudWatch agent using the Docker image version/label 1.231221.0, or later, or the latest. You can find the image at <https://hub.docker.com/r/amazon/cloudwatch-agent>.
- To enable the X-Ray SDK to read cluster name and Region information, the quick setup created a file with the following content, and then applied it with the `kubectl apply -f` command.

```

---
# create role binding for XRay SDK to read config map
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: container-insights-discovery-role
  namespace: amazon-cloudwatch
rules:
- apiGroups:
  - ""
resourceNames:
- cluster-info
resources:
- configmaps
verbs:
- get

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-users-cloudwatch-discovery-role-binding
  namespace: amazon-cloudwatch
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: container-insights-discovery-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group

```

```
name: system:serviceaccounts
```

- The quick setup exposes the CloudWatch agent port that receives X-Ray SDK metrics. The default port is UDP 25888.

```
ports:  
  - containerPort: 25888  
    hostPort: 25888  
    protocol: UDP
```

- The quick setup merges the agent configuration JSON with the X-Ray SDK metrics configuration with the following JSON.

```
{  
  "logs": {  
    "metrics_collected": {  
      "emf": {}  
    }  
  }  
}
```

Deploying the CloudWatch agent and the X-Ray daemon on Amazon EC2

Standard installations of the CloudWatch agent and the X-Ray daemon are sufficient to enable ServiceLens on Amazon EC2, with the addition of the following CloudWatch agent configuration section example. For more information about installing the agent, see [Installing the CloudWatch agent \(p. 483\)](#). For more information about the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

When you configure the CloudWatch agent, include this section in your configuration file:

```
{  
  "logs": {  
    "metrics_collected": {  
      "emf": {}  
    }  
  }  
}
```

For more information about installing the X-Ray daemon, see [X-Ray Daemon Configuration](#).

Using the service map in ServiceLens

This section introduces the service map and helps you learn to navigate it.

To see a service map, you must have installed AWS X-Ray and completed the other ServiceLens deployment steps. For more information, see [Deploying ServiceLens \(p. 269\)](#).

You must also be signed in to an account that has the `AWSXrayReadOnlyAccess` managed policy, as well as permissions that enable you to view the CloudWatch console. For more information, see [How AWS X-Ray Works with IAM](#) and [Using Amazon CloudWatch dashboards \(p. 18\)](#).

To begin using the service map

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **ServiceLens, Service Map**.

A service map appears. It has the following parts:

- The AWS services and your custom applications that you have enabled tracing for are shown as circles or "nodes." The size of each node indicates the relative number of traced requests that are going to that service.
- Edges, or connections between nodes, are shown as lines connecting the nodes. By default, the thickness of a line indicates the relative number of traced requests between those nodes.

You can use the dropdown menu in the top right to choose whether the number of traced requests or the average latency is used for node and edge sizing. You can also select to use constant size for all nodes and edges.

- The entry point to your nodes is shown on the left as a "Client." A "Client" represents both web server traffic and traced API operation requests.
 - A node outlined partially in red, orange, or purple has issues. Some traced requests to these nodes have faults, errors, or throttling. The percentage of the color outline indicates the percentage of traced requests that are having issues.
 - If a node has a triangle with an exclamation point next to it, at least one CloudWatch alarm related to that node is in alarm state.
3. By default, the data in the map is for the most recent 6-hour time window. To change the timeframe of the window, use the controls at the upper right of the screen. The time range to be shown can be up to 6 hours, and can be as much as 30 days in the past.
 4. If you have enabled X-Ray groups, you can filter the map by selecting an X-ray group in the filter.
 5. To view metrics for a node, choose the node. To then see more information about that node, choose **View logs**, **View traces**, or **View dashboard**.
 6. To focus on the incoming and outgoing connections for a node, select the node and choose **View connections** near the top of the service map.
 7. To see a pop-up displaying latency, errors, requests, and alarm summary statistics for a node, pause on that node.
 8. To see latency statistics for an edge connection, pause on the line representing that edge.
 9. To display alarm status for a service, along with line charts for latency, errors, and trace counts, choose that service node on the map.

For more information about this view, see the following procedures.

10. To view the service map as a table, choose **List view** near the top of the screen. In this view, you can filter and sort the nodes and alarms that are displayed on the map.
11. To see a dashboard with metrics for a specific node, select the node and then choose **View dashboard** near the bottom of the screen.

To view traces for a service or application on the service map

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **ServiceLens, Service Map**.
3. Choose the node that represents the service or application that you want to investigate.

CloudWatch displays line charts of latency, errors, and trace counts for that service, along with a summary of alarm status.

Above those charts are options to dive down to logs and traces for the service.

4. To view traces related to the service, choose **View traces**.

The console switches to the **Traces** view, focused on the service that you are investigating. For more information, see [Using the traces view in ServiceLens \(p. 279\)](#).

Using the traces view in ServiceLens

The traces view enables you to view recent X-Ray traces in your application.

To view traces and dive down for more information

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **ServiceLens**, **Traces**.
3. Under **Filter type**, you can choose different criteria to sort the traces by. As you do, CloudWatch displays the success rate and response time of the traces according to your choice. The list of traces displayed at the bottom of the page is narrowed to traces that match your filter. The first 1000 traces that match your filter are retrieved.

The values in the filter table are populated by the traces that the filter returns, and update as you refine your choices in the filter.

Use the **Custom query** option under **Filter type** to add a custom expression as a filter. Custom expressions can contain keywords, unary or binary operators, and comparison values for the keywords. Keywords can correspond to parts of a trace, such as `responsetime`, `duration`, and `status codes`. For more information, see [Filter Expression Syntax](#).

You can focus your filter further by selecting the check box next to a row under **Traces by** and choosing **Add to filter**. The filter value from the selected row is added to the filter.

4. To see a histogram representing the response time distribution of the traces returned by the current filter, and a table displaying each individual trace, scroll to the bottom of the screen.

Choose one of the traces in the table to view detailed information about the trace.

The trace details page displays a timeline that includes each of the segments that comprise the trace. Choose a segment to view additional associated metadata, including errors where applicable. For traces where the logs can be associated, the log lines associated with the trace are displayed under the timeline.

You can then optionally view those log entries in CloudWatch Logs Insights. To do that, choose **View in CloudWatch Logs Insights** and then choose **Run query**.

Some traces do not have associated log entries.

- If you don't see log entries for Amazon EC2 or Amazon EKS, you need to update to the latest version of the X-Ray SDK. For more information, see [Deploying AWS X-Ray \(p. 269\)](#).
- ServiceLens shows log entries for Amazon ECS only if there are 20 or fewer log groups with names that start with `/ecs/`.
- Currently, log entries for Fargate traces are not available.

Using the resource health view in ServiceLens

You can use the resource health view to automatically discover, manage, and visualize the health and performance of hosts across their applications in a single view. You can visualize the health of their hosts

by a performance dimension such as CPU or memory, and slice and dice hundreds of hosts in a single view using filters. You can filter by tags or by use cases, such as hosts in the same Auto Scaling group or hosts that use the same load balancer,

Prerequisites

To make sure that you get the full benefit of the resource health view, check that you have the following prerequisites.

- To see the memory utilization of your hosts and use it as a filter, you must install the CloudWatch agent on your hosts and set it up to send a memory metric to CloudWatch in the default CWAgent namespace. On Linux and macOS instances, the CloudWatch agent must send the `mem_used_percent` metric. On Windows instances, the agent must send the `Memory % Committed Bytes in Use` metric. These metrics are included if you use the wizard to create the CloudWatch agent configuration file and select any of the pre-defined sets of metrics. Metrics collected by the CloudWatch agent are billed as custom metrics. For more information, see [Installing the CloudWatch agent \(p. 483\)](#).

When you use the CloudWatch agent to collect these memory metrics to use with the resource health view, you must include the following section in the CloudWatch agent configuration file. This section contains the default dimension settings and is created by default, so do not change any part of this section to anything different than what is shown in the following example.

```
"append_dimensions": {  
    "ImageId": "${aws:ImageId}",  
    "InstanceId": "${aws:InstanceId}",  
    "InstanceType": "${aws:InstanceType}",  
    "AutoScalingGroupName": "${aws:AutoScalingGroupName}"  
},
```

- To view all the information available in the resource health view, you must be signed in to an account that has the following permissions. If you are signed on with fewer permissions, you can still use the resource health view but some performance data will not be visible.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "autoscaling:Describe*",  
                "cloudwatch:Describe*",  
                "cloudwatch:Describe*",  
                "cloudwatch:Get*",  
                "cloudwatch>List*",  
                "logs:Get*",  
                "logs:Describe*",  
                "sns:Get*",  
                "sns>List*",  
                "ec2:DescribeInstances",  
                "ec2:DescribeInstanceStatus",  
                "ec2:DescribeRegions"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

To view resource health in your account

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **ServiceLens, Resource Health**.

The resource health page appears, showing a square for each host in your account. Each square is colored based on the current status of that host, based on the setting for **Color by**. Host squares with an alarm symbol have one or more alarms currently in ALARM state.

You can see up to 500 hosts in a single view. If you have more hosts in your account, use the filter settings in step 6 of this procedure.

3. To change what criteria is used to show each host's health, choose a setting for **Color by**. You can choose **CPU Utilization**, **Memory Utilization**, or **Status check**. Memory utilization metrics are available only for hosts that are running the CloudWatch agent and have it configured to collect memory metrics and send them to the default CWAgent namespace. For more information, see [Collecting metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent \(p. 482\)](#).
4. To change the thresholds and the colors that are used for the health indicators in the grid, choose the gear icon above the grid.
5. To toggle whether to show alarms in the host grid, choose or clear **Show alarms across all metrics**.
6. To split the hosts in the map into groups, choose a grouping criteria for **Group by**.
7. To narrow the view to fewer hosts, choose a filter criteria for **Filter by**. You can filter by tags and by resource groupings such as Auto Scaling group, instance type, security group, and more.
8. To sort hosts, choose a sorting criteria for **Sort by**. You can sort by status check results, instance state, CPU or memory utilization, and the number of alarms that are in ALARM state.
9. To see more information about a host, choose the square that represents that host. A popup pane appears. To then dive deeper into information about that host, choose **View dashboard** or **View on list**.

ServiceLens troubleshooting

The following sections can help if you're having issues with CloudWatch ServiceLens.

I don't see all my logs

How to configure logs to appear in ServiceLens depends on the service.

- API Gateway logs appear if logging is turned on in API Gateway.
- Amazon ECS and Amazon EKS logs appear if you are using the latest versions of the X-Ray SDK and the CloudWatch agent. For more information, see [Deploying ServiceLens \(p. 269\)](#).
- Lambda logs appear if the request ID is in the log entry. This happens automatically for the situations listed in the following table. For other cases, where the runtime does not automatically include the trace ID, you can manually include the trace ID.

| Runtime | Method | Request ID automatically in log entry? |
|---------|---|--|
| Java | context.getLogger.log
aws-lambda-java-log4j2 | Yes |
| Java | System.out.println | No |

| Runtime | Method | Request ID automatically in log entry? |
|---------|--|--|
| Python | context.log
logging.info/error/log/etc... | Yes |
| Python | print | No |
| Node.js | context.log
console.log/info/error/etc... | Yes |
| dotnet | context.Logger.log
Console.WriteLine() | No |
| Go | fmt.Printf
log.Print | No |
| Ruby | puts | No |

I don't see all my alarms on the service map

ServiceLens shows only the alert icon for a node if any alarms associated with that node are in the ALARM state.

ServiceLens associates alarms with nodes using the following logic:

- If the node represents an AWS service, then all alarms with the namespace associated with that service are associated with the node. For example, a node of type `AWS::Kinesis` is linked with all alarms that are based on metrics in the CloudWatch namespace `AWS/Kinesis`.
- If the node represents an AWS resource, then the alarms on that specific resource are linked. For example, a node of type `AWS::DynamoDB::Table` with the name "MyTable" is linked to all alarms that are based on a metric with the namespace `AWS/DynamoDB` and have the `TableName` dimension set to `MyTable`.
- If the node is of unknown type, which is identified by a dashed border around the name, then no alarms are associated with that node.

I don't see some AWS resources on the service map

For AWS resources to be traced on the service map, the AWS SDK must be captured using the X-Ray SDK. For more information about X-Ray, see [What Is AWS X-Ray](#).

Not every AWS resource is represented by a dedicated node. Some AWS services are represented by a single node for all requests to the service. The following resource types are displayed with a node per resource:

- `AWS::DynamoDB::Table`
- `AWS::Lambda::Function`

Lambda functions are represented by two nodes—one for the Lambda Container, and one for the function. This helps to identify cold start problems with Lambda functions. Lambda container nodes are associated with alarms and dashboards in the same way as Lambda function nodes.

- `AWS::ApiGateway::Stage`

- AWS::SQS::Queue
- AWS::SNS::Topic

There are too many nodes on my service map

Use X-Ray groups to break your map into multiple maps. For more information, see [Using Filter Expressions with Groups](#).

Cross-account cross-Region CloudWatch console

You can add *cross-account* functionality to your CloudWatch console. This functionality provides you with cross-account visibility to your dashboards, alarms, metrics, and automatic dashboards without having to log in and log out of different accounts.

You can then create dashboards that summarize CloudWatch data from multiple AWS accounts and multiple AWS Regions into a single dashboard. You can also create an alarm in one account that watches a metric located in a different account.

Many organizations have their AWS resources deployed in multiple accounts, to provide billing and security boundaries. In this case, we recommend that you designate one or more of your accounts as your monitoring accounts, and build your cross-account dashboards in these accounts.

Cross-account functionality is integrated with AWS Organizations, to help you efficiently build your cross-account dashboards.

Cross-Region functionality

Cross-Region functionality is now built in automatically. You do not need to take any extra steps to be able to display metrics from different Regions in a single account on the same graph or the same dashboard. Cross-Region functionality is not supported for alarms, so you can't create an alarm in one Region that watches a metric in a different Region.

Topics

- [Enabling cross-account functionality in CloudWatch \(p. 284\)](#)
- [\(Optional\) Integrate with AWS Organizations \(p. 287\)](#)
- [Troubleshooting your CloudWatch cross-account setup \(p. 287\)](#)
- [Disabling and cleaning up after using cross-account \(p. 288\)](#)

Enabling cross-account functionality in CloudWatch

To set up cross-account functionality in your CloudWatch console, use the CloudWatch console to set up your sharing accounts and monitoring accounts.

Set up a sharing account

You must enable sharing in each account that will make data available to the monitoring account.

This will grant the read-only permissions that you choose in step 5 to all users that view a cross account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To enable your account to share CloudWatch data with other accounts

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**, then choose **Configure**.

3. Choose **Share data**.
4. For **Sharing**, choose **Specific accounts** and enter the IDs of the accounts that you want to share data with.

Any accounts that you specify here can view your account's CloudWatch data. Specify the IDs only of accounts that you know and trust.
5. For **Permissions**, specify how to share your data with one of the following options:
 - **Provide read-only access to your CloudWatch metrics, dashboards, and alarms.** This option enables the monitoring accounts to create cross-account dashboards that include widgets that contain CloudWatch data from your account.
 - **Include CloudWatch automatic dashboards.** If you select this option, users in the monitoring account can also view the information in this account's automatic dashboards. For more information, see [Getting started with Amazon CloudWatch \(p. 11\)](#).
 - **Include X-Ray read-only access for ServiceLens.** If you select this option, users in the monitoring account can also view the ServiceLens service map and X-Ray trace information in this account. For more information, see [Using ServiceLens to monitor the health of your applications \(p. 268\)](#).
 - **Full read-only access to everything in your account.** This option enables the accounts that you use for sharing to create cross-account dashboards that include widgets that contain CloudWatch data from your account. It also enables those accounts to look deeper into your account and view your account's data in the consoles of other AWS services.
6. Choose **Launch CloudFormation template**.

In the confirmation screen, type **Confirm**, and choose **Launch template**.
7. Select the **I acknowledge...** check box, and choose **Create stack**.

Sharing with an entire organization

Completing the preceding procedure creates an IAM role which enables your account to share data with one account. You can create or edit an IAM role that shares your data with all accounts in an organization. Do this only if you know and trust all accounts in the organization.

This will grant the read-only permissions listed in the policies shown in step 5 of the previous procedure to all users that view a cross-account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To share your CloudWatch account data with all accounts in an organization

1. If you haven't already, complete the preceding procedure to share your data with one AWS account.
2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Roles**.
4. In the list of roles, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.

You see a policy like this:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::123456789012:root"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

```
        }
    ]
```

6. Change the policy to the following, replacing *org-id* with the ID of your organization.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "org-id"
        }
      }
    }
  ]
}
```

7. Choose **Update Trust Policy**.

Set up a monitoring account

Enable each monitoring account if you want to view cross-account CloudWatch data.

When you complete the following procedure, CloudWatch creates a service-linked role that CloudWatch uses in the monitoring account to access data shared from your other accounts. This service-linked role is called **AWSServiceRoleForCloudWatchCrossAccount**. For more information, see [Using service-linked roles for CloudWatch \(p. 922\)](#).

To enable your account to view cross-account CloudWatch data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**, and then, in the **Cross-account cross-region** section, choose **Configure**.
3. Under the **View cross-account cross-region** section, choose **Enable**, and then select the **Show selector in the console** checkbox to enable an account selector to appear in the CloudWatch console when you're graphing a metric or creating an alarm.
4. Under **View cross-account cross-region**, choose one of the following options:
 - **Account Id Input.** This option prompts you to manually input an account ID each time that you want to switch accounts when you view cross-account data.
 - **AWS Organization account selector.** This option causes the accounts that you specified when you completed your cross-account integration with Organizations to appear. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

To do this, you must have first used your organization management account to allow CloudWatch to see a list of accounts in your organization. For more information, see [\(Optional\) Integrate with AWS Organizations \(p. 287\)](#).

- **Custom account selector.** This option prompts you to enter a list of account IDs. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

You can also enter a label for each of these accounts to help you identify them when choosing accounts to view.

The account selector settings that a user makes here are retained only for that user, not for all other users in the monitoring account.

5. Choose **Enable**.

After you complete this setup, you can create cross-account dashboards. For more information, see [Cross-account cross-Region dashboards \(p. 19\)](#).

(Optional) Integrate with AWS Organizations

If you want to integrate cross-account functionality with AWS Organizations, you must make a list of all accounts in the organization available to the monitoring accounts.

To enable cross-account CloudWatch functionality to access a list of all accounts in your organization

1. Sign in to your organization's management account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Settings**, then choose **Configure**.
4. For **Grant permission to view the list of accounts in the organization**, choose **Specific accounts** to be prompted to enter a list of account IDs. The list of accounts in your organization are shared with only the accounts that you specify here.
5. Choose **Share organization account list**.
6. Choose **Launch CloudFormation template**.

In the confirmation screen, type **Confirm**, and choose **Launch template**.

Troubleshooting your CloudWatch cross-account setup

This section contains troubleshooting tips for cross-account, console deployment in CloudWatch.

I am getting access denied errors displaying cross-account data

Check the following:

- Your monitoring account should have a role named **AWSServiceRoleForCloudWatchCrossAccount**. If it does not, you need to create this role. For more information, see [Set Up a Monitoring Account \(p. 286\)](#).
- Each sharing account should have a role named **CloudWatch-CrossAccountSharingRole**. If it does not, you need to create this role. For more information, see [Set Up A Sharing Account \(p. 284\)](#).
- The sharing role must trust the monitoring account.

To confirm that your roles are set up properly for the CloudWatch cross-account console

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

2. In the navigation pane, choose **Roles**.
3. In the list of roles, make sure the needed role exists. In a sharing account, look for **CloudWatch-CrossAccountSharingRole**. In a monitoring account, look for **AWSServiceRoleForCloudWatchCrossAccount**.
4. If you are in a sharing account and **CloudWatch-CrossAccountSharingRole** already exists, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.
6. Confirm that the policy lists either the account ID of the monitoring account, or the organization ID of an organization that contains the monitoring account.

I don't see an account dropdown in the console

First, check that you have created the correct IAM roles, as discussed in the preceding troubleshooting section. If those are set up correctly, make sure that you have enabled this account to view cross-account data, as described in [Enable Your Account to View Cross-Account Data \(p. 286\)](#).

Disabling and cleaning up after using cross-account

To disable cross-account functionality for CloudWatch, follow these steps.

Step 1: Remove the cross-account stacks or roles

The best method is to remove the AWS CloudFormation stacks that were used to enable cross-account functionality.

- In each of the sharing accounts, remove the **CloudWatch-CrossAccountSharingRole** stack.
- If you used AWS Organizations to enable cross-account functionality with all accounts in an organization, remove the **CloudWatch-CrossAccountListAccountsRole** stack in the organization's management account.

If you didn't use the AWS CloudFormation stacks to enable cross-account functionality, do the following:

- In each of the sharing accounts, delete the **CloudWatch-CrossAccountSharingRole** IAM role.
- If you used AWS Organizations to enable cross-account functionality with all accounts in an organization, delete the **CloudWatch-CrossAccountSharing-ListAccountsRole** IAM role in the organization's management account.

Step 2: Remove the service-linked role

In the monitoring account, delete the **AWSServiceRoleForCloudWatchCrossAccount** service-linked IAM role.

Using CloudWatch anomaly detection

When you enable *anomaly detection* for a metric, CloudWatch applies statistical and machine learning algorithms. These algorithms continuously analyze metrics of systems and applications, determine normal baselines, and surface anomalies with minimal user intervention.

The algorithms generate an anomaly detection model. The model generates a range of expected values that represent normal metric behavior.

You can use the model of expected values in two ways:

- Create anomaly detection alarms based on a metric's expected value. These types of alarms don't have a static threshold for determining alarm state. Instead, they compare the metric's value to the expected value based on the anomaly detection model.

You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or both.

For more information, see [Creating a CloudWatch alarm based on anomaly detection \(p. 141\)](#).

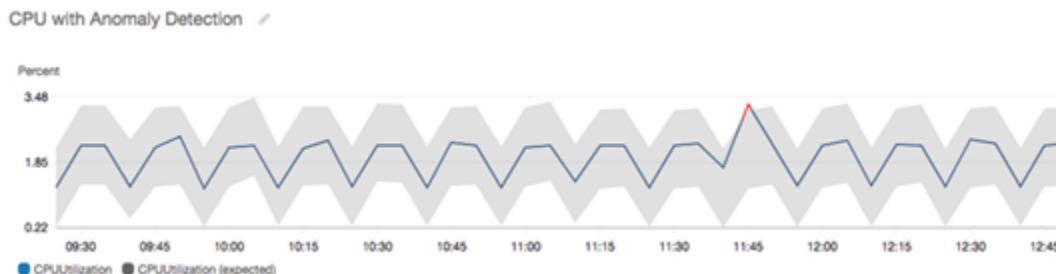
- When viewing a graph of metric data, overlay the expected values onto the graph as a band. This makes it visually clear which values in the graph are out of the normal range. For more information, see [Creating a graph \(p. 85\)](#).

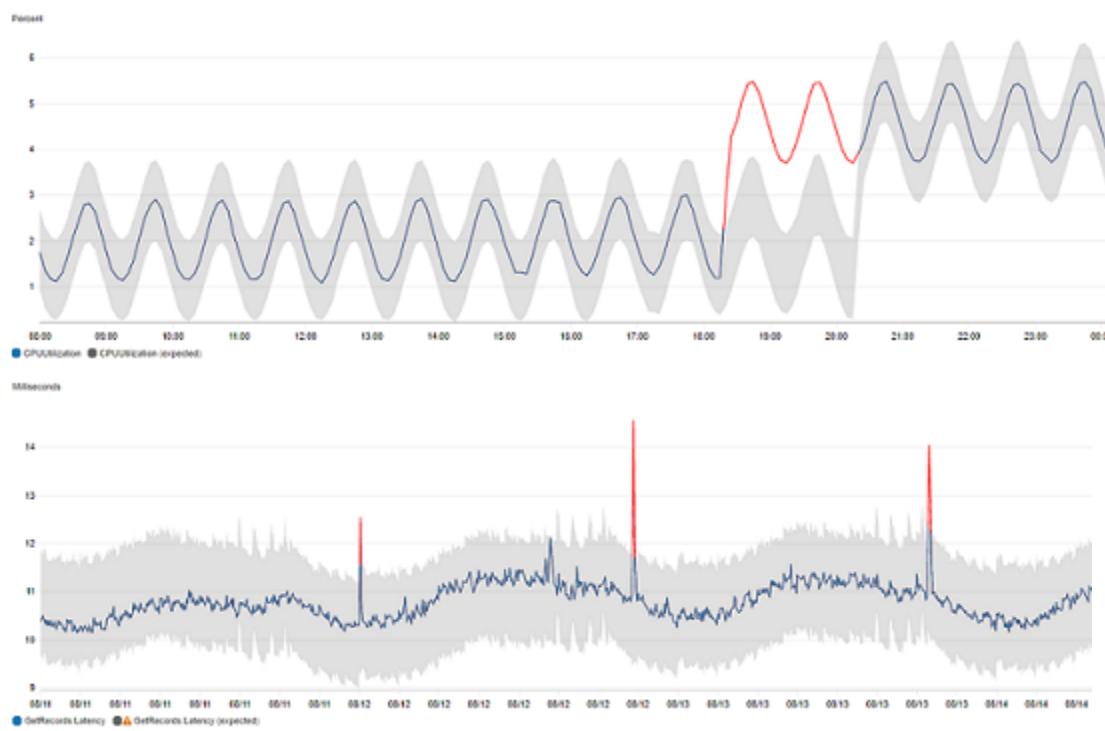
You can enable anomaly detection using the AWS Management Console, the AWS CLI, AWS CloudFormation, or the AWS SDK. You can enable anomaly detection on metrics vended by AWS and also on custom metrics.

You can also retrieve the upper and lower values of the model's band by using the `GetMetricData` API request with the `ANOMALY_DETECTION_BAND` metric math function. For more information, see [GetMetricData](#).

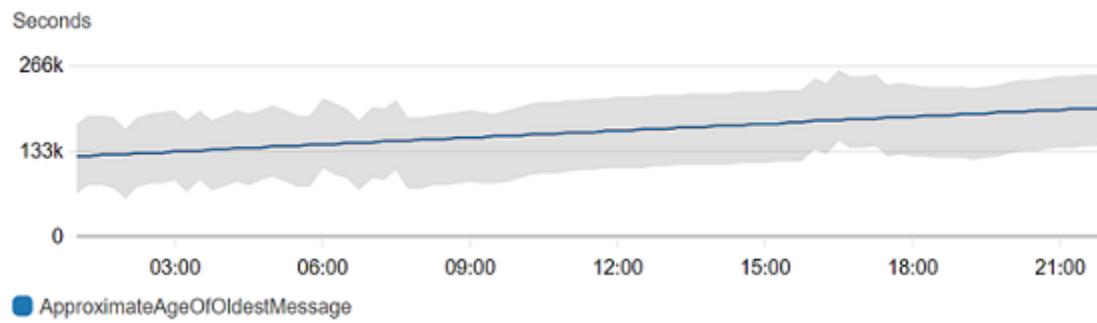
In a graph with anomaly detection, the expected range of values is shown as a gray band. If the metric's actual value goes beyond this band, it is shown as red during that time.

Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes could be hourly, daily, or weekly, as shown in the following examples.

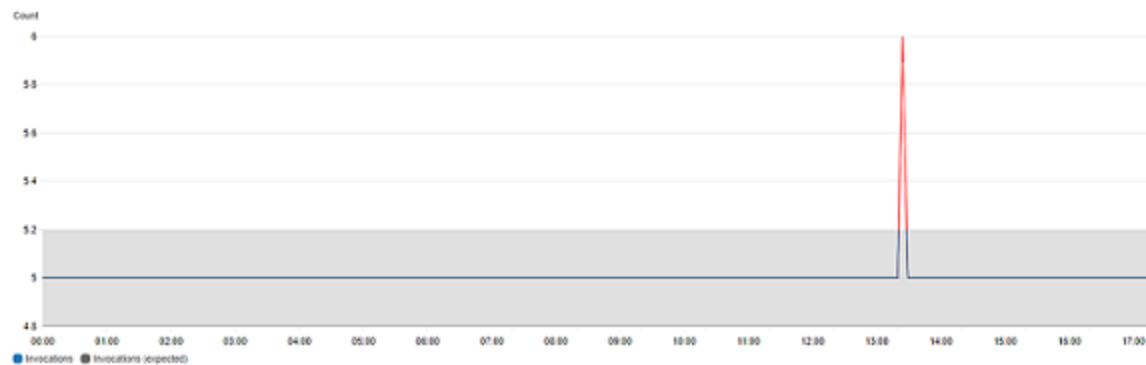




The longer-range trends could be downward or upward.



Anomaly detections also works well with metrics with flat patterns.



How CloudWatch anomaly detection works

When you enable anomaly detection for a metric, CloudWatch applies machine learning algorithms to the metric's past data to create a model of the metric's expected values. The model assesses both trends and hourly, daily, and weekly patterns of the metric. The algorithm trains on up to two weeks of metric data, but you can enable anomaly detection on a metric even if the metric does not have a full two weeks of data.

You specify a value for the anomaly detection threshold that CloudWatch uses along with the model to determine the "normal" range of values for the metric. A higher value for the anomaly detection threshold produces a thicker band of "normal" values.

The machine learning model is specific to a metric and a statistic. For example, if you enable anomaly detection for a metric using the `AVG` statistic, the model is specific to the `AVG` statistic.

When CloudWatch creates a model for many common metrics from AWS services, it ensures that the band doesn't extend outside of logical values. For example, a band for a statistic that can't be negative will never extend below zero, and a band for a percentage metric will stay between 0 and 100.

After you create a model, CloudWatch anomaly detection continually evaluates the model and makes adjustments to it to ensure that it is as accurate as possible. This includes re-training the model to adjust if the metric values evolve over time or have sudden changes, and also includes predictors to improve the models of metrics that are seasonal, spiky, or sparse.

After you enable anomaly detection on a metric, you can choose to exclude specified time periods of the metric from being used to train the model. This way, you can exclude deployments or other unusual events from being used for model training, ensuring the most accurate model is created.

Using anomaly detection models for alarms incurs charges on your AWS account. For more information, see [Amazon CloudWatch Pricing](#).

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on the output metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions. For information about functions that are not supported, see [Using metric math in the Amazon CloudWatch User Guide](#).

You can create anomaly detection models based on metric math expressions similar to how you already create anomaly detection models. From the CloudWatch console, you can apply anomaly detection to metric math expressions and select anomaly detection as a threshold type for these expressions.

Note

Anomaly detection on metric math only can be enabled and edited in the latest version of the metrics user interface. When you create anomaly detectors based on metric math expressions in the new version of the interface, you can view them in the old version, but not edit them.

For information about how to create alarms and models for anomaly detection and metric math, see the following sections:

- [Creating a CloudWatch alarm based on anomaly detection](#)
- [Creating a CloudWatch alarm based on a metric math expression](#)

You also can create, delete, and discover anomaly detection models based on metric math expressions using the CloudWatch API with `PutAnomalyDetector`, `DeleteAnomalyDetector`, and

`DescribeAnomalyDetectors`. For information about these API actions, see the following sections in the *Amazon CloudWatch API Reference*.

- [PutAnomalyDetector](#)
- [DeleteAnomalyDetector](#)
- [DescribeAnomalyDetectors](#)

For information about how anomaly detection alarms are priced, see [Amazon CloudWatch pricing](#).

Using Contributor Insights to analyze high-cardinality data

You can use Contributor Insights to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. This helps you find top talkers and understand who or what is impacting system performance. For example, you can find bad hosts, identify the heaviest network users, or find the URLs that generate the most errors.

You can build your rules from scratch, and when you use the AWS Management Console you can also use sample rules that AWS has created. Rules define the log fields that you want to use to define contributors, such as `IpAddress`. You can also filter the log data to find and analyze the behavior of individual contributors.

CloudWatch also provides built-in rules that you can use to analyze metrics from other AWS services. Currently, built-in rules are available for Amazon DynamoDB.

All rules analyze incoming data in real time.

Note

If you use Contributor Insights, you are charged for each occurrence of a log event that matches a rule. For more information, see [Amazon CloudWatch Pricing](#).

Topics

- [Creating a Contributor Insights rule \(p. 293\)](#)
- [Contributor Insights rule syntax \(p. 296\)](#)
- [Contributor Insights rule examples \(p. 299\)](#)
- [Viewing Contributor Insights reports \(p. 301\)](#)
- [Graphing metrics generated by rules \(p. 302\)](#)
- [Using Contributor Insights built-in rules \(p. 304\)](#)

Creating a Contributor Insights rule

You can create rules to analyze log data. Any logs in JSON or Common Log Format (CLF) can be evaluated. This includes your custom logs that follow one of these formats and logs from AWS services such as Amazon VPC flow logs, Amazon Route 53 DNS query logs, Amazon ECS container logs, and logs from AWS CloudTrail, Amazon SageMaker, Amazon RDS, AWS AppSync and API Gateway.

In a rule, when you specify field names or values, all matching is case sensitive.

You can use built-in sample rules when you create a rule or you can create your own rule from scratch. Contributor Insights includes sample rules for the following types of logs:

- Amazon API Gateway logs
- Amazon Route 53 public DNS query logs
- Amazon Route 53 resolver query logs
- CloudWatch Container Insights logs
- VPC flow logs

Important

When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see [Using condition keys to limit Contributor Insights users' access to log groups \(p. 920\)](#).

To create a rule using a built-in sample rule

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Contributor Insights**.
3. Choose **Create rule**.
4. Choose **Sample rule**, and from **Select sample rule**, select the rule.
5. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, "-", "_", and ":".
6. For **Log group(s)**, select the log groups that you want the rule to monitor. You can select as many as 20 log groups.

To select all log groups with names that start with a certain string, choose **Select by prefix match** and enter the prefix.

If you use **Select by prefix match**, be aware of how many log groups will match your prefix and be analyzed by the rule. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

7. The sample rule has filled out the **Fields**, **Contribution**, **Filters**, and **Aggregate on** fields. You can adjust those values, if you like.
8. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see [Amazon CloudWatch Pricing](#).

Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.

9. Choose **Create**.

To create a rule from scratch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Contributor Insights**.
3. Choose **Create rule**.
4. In the **Create rule** wizard, choose **Custom rule**.
5. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, "-", "_", and ":".
6. You can finish creating the rule by using the wizard or by choosing the **Syntax** tab and specifying your rule syntax manually.

To continue using the wizard, do the following:

- a. For **Log group(s)**, select the log groups that you want the rule to monitor. You can select as many as 20 log groups.

To select all log groups with names that start with a certain string, choose **Select by prefix match** and enter that prefix.

If you use **Select by prefix match**, be aware of how many log groups will match your prefix and be analyzed by the rule. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

- b. For **Log format**, choose **JSON** or **CLF**.
- c. For **Contribution, Key**, enter a contributor type that you want to report on. The report displays the top-N values for this contributor type.

Valid entries are any log field that has values. Examples include **requestId**, **sourceIPAddress**, and **containerID**.

For information about finding the log field names for the logs in a certain log group, see [Finding Log Fields \(p. 296\)](#).

Keys larger than 1 KB are truncated to 1KB.

- d. (Optional) Add more values for **Contribution, Key**. You can include as many as four keys in a rule. If you enter more than one key, the contributors in the report are defined by unique value combinations of the keys. For example, if you specify three keys, each unique combination of values for the three keys is counted as a unique contributor.
- e. (Optional) If you want to rank contributors by the value of a numerical log field, instead of by number of occurrences in log events, use the **Contribution, Value** field. Enter the name of a numerical log field that you want to sum to determine contributor ranking. For example, if you want to find the source IP addresses that are sending the most bytes over the network, you would add **bytes** as the value, assuming that **bytes** is the correct keyword for that field in the log events.
- f. (Optional) If you want to add a filter that narrows the scope of your results, choose **Add filter**. For **Match**, enter the name of the log field that you want to filter on. For **Condition**, choose a comparison operator, and enter a value that you want to filter for. You can use * as a wildcard in the value.

For example, if you want to analyze only the log events in an Apache log that contain errors, for **Match**, you would specify **RESPONSE_CODE**, for **Condition**, you would specify **EqualTo**, and then you would enter **5**** as the value to filter for.

You can add as many as four filters in a rule. Multiple filters are joined by AND logic, so only log events that match all filters are evaluated.

Note

Arrays that follow comparison operators, such as **In**, **NotIn**, or **StartsWith**, can include as many as 10 string values. For more information about the Contributor Insights rules syntax, see [Contributor Insights rule syntax \(p. 296\)](#).

- g. For **Aggregate on**, choose **Count** or **Sum**. Choosing **Count** causes the contributor ranking to be based on the number of occurrences. Choosing **Sum** causes the ranking to be based on the aggregated sum of the values of the field that you specify for **Contribution, Value**.
7. To enter your rule as a JSON object instead of using the wizard, do the following:
 - a. Choose the **Syntax** tab.
 - b. In **Rule body**, enter the JSON object for your rule. For information about rule syntax, see [Contributor Insights rule syntax \(p. 296\)](#).
8. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see [Amazon CloudWatch Pricing](#).

Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.

9. Choose **Create**.

You can disable, enable, or delete rules that you have created.

To enable, disable, or delete a rule in Contributor Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Contributor Insights**.
3. In the list of rules, select the check box next to a single rule.
Built-in rules are created by AWS services and can't be edited, disabled, or deleted.
4. Choose **Actions**, and then choose the option you want.

Finding log fields

When you create a rule, you need to know the names of fields in the log entries in a log group.

To find the log fields in a log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, under **Logs**, choose **Insights**.
3. Above the query editor, select one or more log groups to query.

When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in the right pane in **Discovered fields**.

Contributor Insights rule syntax

This section explains the syntax for Contributor Insights rules. Use this syntax only when you are creating a rule by entering a JSON block. If you use the wizard to create a rule, you don't need to know the syntax. For more information about creating rules using the wizard, see [Creating a Contributor Insights rule \(p. 293\)](#).

All matching of rules to log event field names and values is case sensitive.

The following example illustrates the syntax for JSON logs.

```
{  
    "Schema": {  
        "Name": "CloudWatchLogRule",  
        "Version": 1  
    },  
    "LogGroupNames": [  
        "API-Gateway-Access-Logs*",  
        "Log-group-name2"  
    ],  
    "LogFormat": "JSON",  
    "Contribution": {  
        "Keys": [  
            "$.ip"  
        ],  
        "ValueOf": "$.requestBytes",  
        "Filters": [  
            {  
                "Match": "$.httpMethod",  
                "Value": "GET"  
            }  
        ]  
    }  
}
```

```
        "In": [
            "PUT"
        ]
    },
    "AggregateOn": "Sum"
}
```

Fields in Contributor Insights rules

Schema

The value of Schema for a rule that analyzes CloudWatch Logs data must always be { "Name" : "CloudWatchLogRule", "Version": 1}

LogGroupNames

An array of strings. For each element in the array, you can optionally use * at the end of a string to include all log groups with names that start with that prefix.

Be careful about using wildcards with log group names. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

LogFormat

Valid values are **JSON** and **CLF**.

Contribution

This object includes a **Keys** array with as many as four members, optionally a single **ValueOf**, and optionally an array of as many as four **Filters**.

Keys

An array of up to four log fields that are used as dimensions to classify contributors. If you enter more than one key, each unique combination of values for the keys is counted as a unique contributor. The fields must be specified using JSON property format notation.

ValueOf

(Optional) Specify this only when you are specifying **Sum** as the value of **AggregateOn**. **ValueOf** specifies a log field with numerical values. In this type of rule, the contributors are ranked by their sum of the value of this field, instead of their number of occurrences in the log entries. For example, if you want to sort contributors by their total **BytesSent** over a period, you would set **ValueOf** to **BytesSent** and specify **Sum** for **AggregateOn**.

Filters

(Optional) Specifies an array of as many as four filters to narrow the log events that are included in the report. If you specify multiple filters, Contributor Insights evaluates them with a logical AND operator. You can use this to filter out irrelevant log events in your search or you can use it to select a single contributor to analyze their behavior.

Each member in the array must include a **Match** field and a field indicating the type of matching operator to use.

The **Match** field specifies a log field to evaluate in the filter. The log field is specified using JSON property format notation.

The matching operator field must be one of the following: **In**, **NotIn**, **StartsWith**, **Greater Than**, **LessThan**, **EqualTo**, **NotEqualTo**, or **IsPresent**. If the operator field is **In**, **NotIn**, or **StartsWith**, it is followed by an array of string values to check for. Contributor Insights evaluates the array of string values with an OR operator. The array can include as many as 10 string values.

If the operator field is `GreaterThan`, `LessThan`, `EqualTo`, or `NotEqualTo`, it is followed by a single numerical value to compare with.

If the operator field is `IsPresent`, it is followed by either `true` or `false`. This operator matches log events based on whether the specified log field is present in the log event. The `isPresent` works only with values in the leaf node of JSON properties. For example, a filter that looks for matches to `c-count` does not evaluate a log event with a value of `details.c-count.c1`.

See the following four filter examples:

```
{"Match": "$.httpMethod", "In": [ "PUT", ] }
{"Match": "$.StatusCode", "EqualTo": 200 }
{"Match": "$.BytesReceived", "GreaterThanOrEqualTo": 10000}
{"Match": "$.eventSource", "StartsWith": [ "ec2", "ecs" ] }
```

AggregateOn

Valid values are `Count` and `Sum`. Specifies whether to aggregate the report based on a count of occurrences or a sum of the values of the field that is specified in the `ValueOf` field.

JSON property format notation

The `Keys`, `ValueOf`, and `Match` fields follow JSON property format with dot notation, where `$` represents the root of the JSON object. This is followed by a period and then an alphanumeric string with the name of the subproperty. Multiple property levels are supported.

The following list illustrates valid examples of JSON property format:

```
$.userAgent
$.endpoints[0]
$.users[1].name
$.requestParameters.instanceId
```

Additional field in rules for CLF logs

Common Log Format (CLF) log events do not have names for the fields like JSON does. To provide the fields to use for Contributor Insights rules, a CLF log event can be treated as array with an index starting from 1. You can specify the first field as `"1"`, the second field as `"2"`, and so on.

To make a rule for a CLF log easier to read, you can use `Fields`. This enables you to provide a naming alias for CLF field locations. For example, you can specify that the location `"4"` is an IP address. Once specified, `IpAddress` can be used as property in the `Keys`, `ValueOf`, and `Filters` in the rule.

The following is an example of a rule for a CLF log that uses the `Fields` field.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "API-Gateway-Access-Logs*"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "4": "IpAddress",
    "7": "StatusCode"
  },
  "Contribution": {
```

```

    "Keys": [
        "IpAddress"
    ],
    "Filters": [
        {
            "Match": "StatusCode",
            "EqualTo": 200
        }
    ]
},
"AggregateOn": "Count"
}

```

Contributor Insights rule examples

This section contains examples that illustrate use cases for Contributor Insights rules.

VPC Flow Logs: Byte transfers by source and destination IP address

```

{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "srcaddr",
        "5": "dstaddr",
        "10": "bytes"
    },
    "Contribution": {
        "Keys": [
            "srcaddr",
            "dstaddr"
        ],
        "ValueOf": "bytes",
        "Filters": []
    },
    "AggregateOn": "Sum"
}

```

VPC Flow Logs: Highest number of HTTPS requests

```

{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "5": "destination address",
        "7": "destination port",
        "9": "packet count"
    },
    "Contribution": {

```

```

        "Keys": [
            "destination address"
        ],
        "ValueOf": "packet count",
        "Filters": [
            {
                "Match": "destination port",
                "EqualTo": 443
            }
        ]
    },
    "AggregateOn": "Sum"
}

```

VPC Flow Logs: Rejected TCP connections

```

{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}

```

Route 53 NXDomain responses by source address

```

{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {

```

```
        "Match": "$.rcode",
        "StartsWith": [
            "NXDOMAIN"
        ]
    ],
    "Keys": [
        "$.srcaddr"
    ]
},
"LogFormat": "JSON",
"LogGroupNames": [
    "<loggroupname>"
]
}
```

Route 53 resolver queries by domain name

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_name"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 resolver queries by query type and source address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_type",
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Viewing Contributor Insights reports

To view graphs of report data and a ranked list of contributors found by your rules, follow these steps.

To view your rule reports

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Contributor Insights**.

3. In the list of rules, choose the name of a rule.

The graph displays the results of the rule over the last three hours. The table under the graph shows the top 10 contributors.

4. To change the number of contributors shown in the table, choose **Top 10 contributors** at the top of the graph.
5. To filter the graph to show only the results from a single contributor, choose that contributor in the table legend. To again show all contributors, choose that same contributor again in the legend.
6. To change the time range shown in the report, choose **15m, 30m, 1h, 2h, 3h, or custom** at the top of the graph.

The maximum time range for the report is 24 hours, but you can choose a 24-hour window that occurred up to 15 days ago. To choose a time window in the past, choose **custom, absolute**, and then specify your time window.

7. To change the length of the time period used for the aggregation and ranking of contributors, choose **period** at the top of the graph. Viewing a longer time period generally shows a smoother report with few spikes. Choosing a shorter time period is more likely to display spikes.
8. To add this graph to a CloudWatch dashboard, choose **Add to dashboard**.
9. To open the CloudWatch Logs Insights query window, with the log groups in this report already loaded in the query box, choose **View logs**.

10. To export the report data to your clipboard or a CSV file, choose **Export**.

Graphing metrics generated by rules

Contributor Insights provides a metric math function, `INSIGHT_RULE_METRIC`. You can use this function to add data from a Contributor Insights report to a graph in the **Metrics** tab of the CloudWatch console. You can also set an alarm based on this math function. For more information about metric math functions, see [Using metric math \(p. 97\)](#)

To use this metric math function, you must be signed in to an account that has both the `cloudwatch:GetMetricData` and `cloudwatch:GetInsightRuleReport` permissions.

The syntax is `INSIGHT_RULE_METRIC(ruleName, metricName)`. `ruleName` is the name of a Contributor Insights rule. `metricName` is one of the values in the following list. The value of `metricName` determines which type of data the math function returns.

- `UniqueContributors` — the number of unique contributors for each data point.
- `MaxContributorValue` — the value of the top contributor for each data point. The identity of the contributor might change for each data point in the graph.

If this rule aggregates by `Count`, the top contributor for each data point is the contributor with the most occurrences in that period. If the rule aggregates by `Sum`, the top contributor is the contributor with the greatest sum in the log field specified by the rule's `Value` during that period.

- `SampleCount` — the number of data points matched by the rule.
- `Sum` — the sum of the values from all contributors during the time period represented by that data point.
- `Minimum` — the minimum value from a single observation during the time period represented by that data point.

- Maximum — the maximum value from a single observation during the time period represented by that data point.
- Average — the average value from all contributors during the time period represented by that data point.

Setting an alarm on Contributor Insights metric data

Using the function `INSIGHT_RULE_METRIC`, you can set alarms on metrics that Contributor Insights generates. For example, you can create an alarm that's based on the percentage of rejected transmission control protocol (TCP) connections. To get started with this type of alarm, you can create rules like the ones shown in the following two examples:

Example rule: "RejectedConnectionsRule"

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "3": "interfaceID",
    "4": "sourceAddress",
    "8": "protocol",
    "13": "action"
  },
  "Contribution": {
    "Keys": [
      "interfaceID",
      "sourceAddress"
    ],
    "Filters": [
      {
        "Match": "protocol",
        "EqualTo": 6
      },
      {
        "Match": "action",
        "In": [
          "REJECT"
        ]
      }
    ]
  },
  "AggregateOn": "Sum"
}
```

Example rule: "TotalConnectionsRule"

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
}
```

```
"LogFormat": "CLF",
"Fields": {
    "3": "interfaceID",
    "4": "sourceAddress",
    "8": "protocol",
    "13": "action"
},
"Contribution": {
    "Keys": [
        "interfaceID",
        "sourceAddress"
    ],
    "Filters": [
        {
            "Match": "protocol",
            "EqualTo": 6
        }
    ]
},
"AggregateOn": "Sum"
}
```

After you create your rules, you can select the **Metrics** tab in the CloudWatch Console, where you can use the following example metric math expressions to graph the data that Contributor Insights reports:

Example: Metric math expressions

```
e1 INSIGHT_RULE_METRIC("RejectedConnectionsRule", "Sum")
e2 INSIGHT_RULE_METRIC("TotalConnectionsRule", "Sum")
e3 (e1/e2)*100
```

In the example, the metric math expression e3 returns all of the rejected TCP connections. If you want to be notified when 20 percent of the TCP connections are rejected, you can modify the expression by changing the threshold from 100 to 20.

Note

You can set an alarm on a metric that you're monitoring from the **Metrics** section. While on the **Graphed metrics** tab, you can select the **Create alarm** icon under the **Actions** column. The **Create alarm** icon looks like a bell.

For more information about graphing metrics and using metric math functions, see the following section: [Adding a math expression to a CloudWatch graph \(p. 97\)](#).

Using Contributor Insights built-in rules

Other AWS services create built-in Contributor Insights rules that evaluate metrics from those AWS services. Currently, Amazon DynamoDB supports built-in rules.

For more information, see [Contributor Insights for Amazon DynamoDB](#).

Using Container Insights

Use CloudWatch Container Insights to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. Container Insights is available for Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Kubernetes platforms on Amazon EC2. Container Insights supports collecting metrics from clusters deployed on Fargate for both Amazon ECS and Amazon EKS.

CloudWatch automatically collects metrics for many resources, such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects.

Container Insights collects data as *performance log events* using [embedded metric format \(p. 761\)](#). These performance log events are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates aggregated metrics at the cluster, node, pod, task, and service level as CloudWatch metrics. The metrics that Container Insights collects are available in CloudWatch automatic dashboards, and also viewable in the **Metrics** section of the CloudWatch console.

CloudWatch does not automatically create all possible metrics from the log data, to help you manage your Container Insights costs. However, you can view additional metrics and additional levels of granularity by using CloudWatch Logs Insights to analyze the raw performance log events.

Metrics collected by Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

In Amazon EKS and Kubernetes, Container Insights uses a containerized version of the CloudWatch agent to discover all of the running containers in a cluster. It then collects performance data at every layer of the performance stack.

Container Insights supports encryption with the customer master key (CMK) for the logs and metrics that it collects. To enable this encryption, you must manually enable KMS encryption for the log group that receives Container Insights data. This results in Container Insights encrypting this data using the provided CMK. Only symmetric CMKs are supported. Do not use asymmetric CMKs to encrypt your log groups.

For more information, see [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#).

Supported platforms

Container Insights is available for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2 instances.

- For Amazon ECS, Container Insights collects metrics at the cluster, task and service levels on both Linux and Windows Server instances. It can collect metrics at the instance-level only on Linux instances.

For Amazon ECS, network metrics are available only for containers in `bridge` network mode and `awsvpc` network mode. They are not available for containers in `host` network mode.

- For Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2 instances, Container Insights is supported only on Linux instances.
- Currently, Container Insights isn't supported in AWS Batch.

CloudWatch agent container image

Amazon provides a CloudWatch agent container image on Amazon Elastic Container Registry. For more information, see [cloudwatch-agent](#) on Amazon ECR.

Supported Regions

Container Insights for Amazon ECS is supported in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Mumbai)
- Asia Pacific (Hong Kong)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Jakarta)
- Asia Pacific (Sydney)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)
- China (Beijing)
- China (Ningxia)

Supported Regions for Amazon EKS and Kubernetes

Container Insights for Amazon EKS and Kubernetes is supported in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)

- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Setting up Container Insights

The Container Insights setup process is different for Amazon ECS and Amazon EKS and Kubernetes.

Topics

- [Setting up Container Insights on Amazon ECS \(p. 307\)](#)
- [Setting up Container Insights on Amazon EKS and Kubernetes \(p. 320\)](#)

Setting up Container Insights on Amazon ECS

You can use one or both of the following options to enable Container Insights on Amazon ECS clusters:

- Use the AWS Management Console or the AWS CLI to start collecting cluster-level, task-level, and service-level metrics.
- Deploy the CloudWatch agent as a DaemonSet to start collecting of instance-level metrics on clusters that are hosted on Amazon EC2 instances.

Topics

- [Setting up Container Insights on Amazon ECS for cluster- and service-level metrics \(p. 307\)](#)
- [Setting up Container Insights on Amazon ECS using AWS Distro for OpenTelemetry \(p. 309\)](#)
- [Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS \(p. 311\)](#)
- [Deploying the AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters \(p. 317\)](#)
- [Set up Firelens to send logs to CloudWatch Logs \(p. 319\)](#)

Setting up Container Insights on Amazon ECS for cluster- and service-level metrics

You can enable Container Insights on new and existing Amazon ECS clusters. Container Insights collects metrics at the cluster, task, and service levels. For existing clusters, you use the AWS CLI. For new clusters, use either the Amazon ECS console or the AWS CLI.

If you're using Amazon ECS on an Amazon EC2 instance, and you want to collect network and storage metrics from Container Insights, launch that instance using an AMI that includes Amazon ECS agent version 1.29. For information about updating your agent version, see [Updating the Amazon ECS Container Agent](#)

You can use the AWS CLI to set account-level permission to enable Container Insights for any new Amazon ECS clusters created in your account. To do so, enter the following command.

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

Setting up Container Insights on existing Amazon ECS clusters

To enable Container Insights on an existing Amazon ECS cluster, enter the following command. You must be running version 1.16.200 or later of the AWS CLI for the following command to work.

```
aws ecs update-cluster-settings --cluster myCICluster --settings  
name=containerInsights,value=enabled
```

Setting up Container Insights on new Amazon ECS clusters

There are two ways to enable Container Insights on new Amazon ECS clusters. You can configure Amazon ECS so that all new clusters are enabled for Container Insights by default. Otherwise, you can enable a new cluster when you create it.

Using the AWS Management Console

You can enable Container Insights on all new clusters by default, or on an individual cluster as you create it.

To enable Container Insights on all new clusters by default

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Account Settings**.
3. Select the check box at the bottom of the page to enable the Container Insights default.

If you haven't used the preceding procedure to enable Container Insights on all new clusters by default, use the following steps to create a cluster with Container Insights enabled.

To create a cluster with Container Insights enabled

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters**.
3. Choose **Create cluster**.
4. On the next page, do the following:
 - a. Name your cluster.
 - b. If you don't have a VPC already, select the check box to create one. You can use the default values for the VPC.
 - c. Fill out all other needed information, including instance type.
 - d. Select **Enabled Container Insights**.
 - e. Choose **Create**.

You can now create task definitions, run tasks, and launch services in the cluster. For more information, see the following:

- [Creating a task definition](#)
- [Running tasks](#)
- [Creating a service](#)

Setting up Container Insights on new Amazon ECS clusters using the AWS CLI

To enable Container Insights on all new clusters by default, enter the following command.

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

If you didn't use the preceding command to enable Container Insights on all new clusters by default, enter the following command to create a new cluster with Container Insights enabled. You must be running version 1.16.200 or later of the AWS CLI for the following command to work.

```
aws ecs create-cluster --cluster-name myCICluster --settings  
"name=containerInsights,value=enabled"
```

Disabling Container Insights on Amazon ECS clusters

To disable Container Insights on an existing Amazon ECS cluster, enter the following command.

```
aws ecs update-cluster-settings --cluster myCICluster --settings  
name=containerInsights,value=disabled
```

Setting up Container Insights on Amazon ECS using AWS Distro for OpenTelemetry

Use this section if you want to use AWS Distro for OpenTelemetry to set up CloudWatch Container Insights on an Amazon ECS cluster. For more information about AWS Distro for Open Telemetry, see [AWS Distro for OpenTelemetry](#).

These steps assume that you already have a cluster running Amazon ECS. For more information about using AWS Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service](#).

Step 1: Create a task role

The first step is creating a task role in the cluster that the AWS OpenTelemetry Collector will use.

To create a task role for AWS Distro for OpenTelemetry

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab and copy in the following policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:PutLogEvents",  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs:DescribeLogStreams",  
                "logs:GetLogEvents",  
                "logs:ListLogGroups",  
                "logs:ListLogStreams",  
                "logs:PutMetricFilter",  
                "logs:PutRetentionPolicy",  
                "logs:TestMetricFilter"  
            ]  
        }  
    ]  
}
```

```
        "logs:DescribeLogGroups",
        "ssm:GetParameters"
    ],
    "Resource": "*"
}
]
```

4. Choose **Review policy**.
5. For name, enter **AWS Distro OpenTelemetry Policy**, and then choose **Create policy**.
6. In the left navigation pane, choose **Roles** and then choose **Create role**.
7. In the list of services, choose **Elastic Container Service**.
8. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
9. In the list of policies, search for **AWS Distro OpenTelemetry Policy**.
10. Select the check box next to **AWS Distro OpenTelemetry Policy**.
11. Choose **Next: Tags** and then choose **Next: Review**.
12. For **Role name** enter **AWS OpenTelemetry Task Role** and then choose **Create role**.

Step 2: Create a task execution role

The next step is creating a task execution role for the AWS OpenTelemetry Collector.

To create a task execution role for AWS Distro for OpenTelemetry

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles** and then choose **Create role**.
3. In the list of services, choose **Elastic Container Service**.
4. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
5. In the list of policies, search for **AmazonECSTaskExecutionRolePolicy** and then select the check box next to **AmazonECSTaskExecutionRolePolicy**.
6. In the list of policies, search for **CloudWatchLogsFullAccess** and then select the check box next to **CloudWatchLogsFullAccess**.
7. In the list of policies, search for **AmazonSSMReadOnlyAccess** and then select the check box next to **AmazonSSMReadOnlyAccess**.
8. Choose **Next: Tags** and then choose **Next: Review**.
9. For **Role name** enter **AWS OpenTelemetry Task Execution Role** and then choose **Create role**.

Step 3: Create a task definition

The next step is creating a task definition.

To create a task definition for AWS Distro for OpenTelemetry

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the left navigation pane, choose **Task Definitions** and then choose **Create new Task Definition**.
3. Select either **FARGATE** or **EC2** and then choose **Next step**.
4. Enter a task definition name such as **aws-otel**.
5. For **Task Role**, select **AWS OpenTelemetry Task Role** which you created earlier.
6. For **Task execution role**, select **AWS OpenTelemetry Task Execution Role** which you created earlier.
7. Fill in the **Task memory** and **Task CPU**.
8. Under **Container Definitions**, choose **Add container**.

9. For **Container name**, enter `aws-otel-collector`. For **Image**, enter `public.ecr.aws/aws-observability/aws-otel-collector`.
 10. Under **ENVIRONMENT**, for **Command** enter `--config=/etc/ecs/container-insights/otel-task-metrics-config.yaml`
- This YAML file is included in the Docker image, and includes the configuration to consume container metrics.
11. If you're using the EC2 launch type, enter a port mapping of 55680 for TCP.
 12. Finish the steps for adding the container.

For more information about using the AWS OpenTelemetry collector with Amazon ECS, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service](#).

Step 4: Run the task

The final step is running the task that you've created.

To run the task for AWS Distro for OpenTelemetry

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the left navigation pane, choose **Task Definitions** and then select the task that you just created.
3. Choose **Actions, Run Task**.

Next, you can check for the new metrics in the CloudWatch console.

4. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
5. In the left navigation pane, choose **Metrics**.

You should see a **ECS/ContainerInsights** namespace. Choose that namespace and you should see eight metrics.

Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS

To deploy the CloudWatch agent to collect instance-level metrics from Amazon ECS clusters that are hosted on EC2 instance, use a quick start setup with a default configuration, or install the agent manually to be able to customize it.

Both methods require that you already have at least one Amazon ECS cluster deployed with an EC2 launch type. These methods also assume that you have the AWS CLI installed. Additionally, to run the commands in the following procedures, you must be logged on to an account or role that has the **IAMFullAccess** and **AmazonECS_FullAccess** policies.

Topics

- [Quick setup using AWS CloudFormation \(p. 311\)](#)
- [Manual and custom setup \(p. 313\)](#)

Quick setup using AWS CloudFormation

To use the quick setup, enter the following command to use AWS CloudFormation to install the agent. Replace `cluster-name` and `cluster-region` with the name and Region of your Amazon ECS cluster.

This command creates the IAM roles **CWAgentECSTaskRole** and **CWAgentECSExecutionRole**. If these roles already exist in your account, use `ParameterKey=CreateIAMRoles,ParameterValue=False`

instead of `ParameterKey=CreateIAMRoles,ParameterValue=True` when you enter the command. Otherwise, the command will fail.

Note

The following setup step pulls the container image from Docker Hub as an anonymous user by default. This pull may be subject to a rate limit. For more information, see [CloudWatch agent container image \(p. 306\)](#).

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
    container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/
    cwagent-ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
    \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
        ParameterKey=CreateIAMRoles,ParameterValue=True \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

(Alternative) Using your own IAM roles

If you want to use your own custom ECS task role and ECS task execution role instead of the **CWAgentECSTaskRole** and **CWAgentECSExecutionRole** roles, first make sure that the role to be used as the ECS task role has **CloudWatchAgentServerPolicy** attached. Also, make sure that the role to be used as the ECS task execution role has both the **CloudWatchAgentServerPolicy** and **AmazonECSTaskExecutionRolePolicy** policies attached. Then enter the following command. In the command, replace `task-role-arn` with the ARN of your custom ECS task role, and replace `execution-role-arn` with the ARN of your custom ECS task execution role.

```
ClusterName=cluster-name
Region=cluster-region
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
    container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/
    cwagent-ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
    \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
        ParameterKey=TaskRoleArn,ParameterValue=${TaskRoleArn} \
        ParameterKey=ExecutionRoleArn,ParameterValue=${ExecutionRoleArn} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

Troubleshooting the quick setup

To check the status of the AWS CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stacks --stack-name CWAgentECS-$ClusterName-$Region --region
$Region
```

If you see the `StackStatus` is other than `CREATE_COMPLETE` or `CREATE_IN_PROGRESS`, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
```

```
aws cloudformation describe-stack-events --stack-name CWAgentECS-$ClusterName-$Region --region $Region
```

To check the status of the cwagent daemon service, enter the following command. In the output, you should see that the runningCount is equal to the desiredCount in the deployment section. If it isn't equal, check the failures section in the output.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services cwagent-daemon-service --cluster $ClusterName --region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/ecs/ecs-cwagent-daemon-service** log group.

Deleting the AWS CloudFormation stack for the CloudWatch agent

If you need to delete the AWS CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation delete-stack --stack-name CWAgentECS-${ClusterName}-${Region} --region ${Region}
```

Manual and custom setup

Follow the steps in this section to manually deploy the CloudWatch agent to collect instance-level metrics from your Amazon ECS clusters that are hosted on EC2 instances.

Necessary IAM roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see [IAM roles for Tasks](#) and [Amazon ECS Task Execution Role](#).

- An *ECS task role*, which is used by the CloudWatch agent to publish metrics. If this role already exists, you must make sure it has the `CloudWatchAgentServerPolicy` policy attached.
- An *ECS task execution role*, which is used by Amazon ECS agent to launch the CloudWatch agent. If this role already exists, you must make sure it has the `AmazonECSTaskExecutionRolePolicy` and `CloudWatchAgentServerPolicy` policies attached.

If you do not already have these roles, you can use the following commands to create them and attach the necessary policies. This first command creates the ECS task role.

```
aws iam create-role --role-name CWAgentECSTaskRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com\"}, \"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as `"TaskRoleArn"`. You'll need to use it later when you create the task definition. Then enter the following command to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
    --role-name CWAgentECSTaskRole
```

This next command creates the ECS task execution role.

```
aws iam create-role --role-name CWAgentECSExecutionRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com\"}, \"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of Arn from the command output as "ExecutionRoleArn". You'll need to use it later when you create the task definition. Then enter the following commands to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
    \
    --role-name CWAgentECSExecutionRole

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy \
    \
    --role-name CWAgentECSExecutionRole
```

Create the task definition and launch the daemon service

Create a task definition and use it to launch the CloudWatch agent as a daemon service. To create the task definition, enter the following command. In the first lines, replace the placeholders with the actual values for your deployment. *logs-region* is the Region where CloudWatch Logs is located, and *cluster-region* is the Region where your cluster is located. *task-role-arn* is the Arn of the ECS task role that you are using, and *execution-role-arn* is the Arn of the ECS task execution role.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

Then run the following command to launch the daemon service. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

If you see this error message, An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent, you have already created a daemon service named cwagent-daemon-service. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs delete-service \
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
```

```
--region ${Region} \
--force
```

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the CloudWatch agent in your Amazon ECS clusters that are hosted on EC2 instances. These options are as follows:

- `metrics_collection_interval` – How often in seconds that the CloudWatch agent collects metrics. The default is 60. The range is 1–172,000.
- `endpoint_override` – (Optional) Specifies a different endpoint to send logs to. You might want to do this if you're publishing from a cluster in a VPC and you want the logs data to go to a VPC endpoint.

The value of `endpoint_override` must be a string that is a URL.

- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5 seconds.
- `region` – By default, the agent publishes metrics to the same Region where the Amazon ECS container instance is located. To override this, you can specify a different Region here. For example, "region" : "us-east-1"

The following is an example of a customized configuration:

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "ecs": {
        "metrics_collection_interval": 30
      }
    },
    "force_flush_interval": 5
  }
}
```

To customize your CloudWatch agent configuration in your Amazon ECS containers

1. Make sure that the **AmazonSSMReadOnlyAccess** policy is attached to your Amazon ECS Task Execution role. You can enter the following command to do so. This example assumes that your Amazon ECS Task Execution role is **CWAgentECSExecutionRole**. If you are using a different role, substitute that role name in the following command.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess
 \
 --role-name CWAgentECSExecutionRole
```

2. Create the customized configuration file similar to the preceding example. Name this file `/tmp/ecs-cwagent-daemon-config.json`.
3. Run the following command to put this configuration into the Parameter Store. Replace **cluster-region** with the Region of your Amazon ECS cluster. To run this command, you must be logged on to a user or role that has the **AmazonSSMFullAccess** policy.

```
Region=cluster-region
aws ssm put-parameter \
```

```
--name "ecs-cwagent-daemon-service" \
--type "String" \
--value "`cat /tmp/ecs-cwagent-daemon-config.json`" \
--region $Region
```

4. Download the task definition file to a local file, such as `/tmp/cwagent-ecs-instance-metric.json`

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json -o /tmp/cwagent-ecs-instance-metric.json
```

5. Modify the task definition file. Remove the following section:

```
"environment": [
    {
        "name": "USE_DEFAULT_CONFIG",
        "value": "True"
    }
],
```

Replace that section with the following:

```
"secrets": [
    {
        "name": "CW_CONFIG_CONTENT",
        "valueFrom": "ecs-cwagent-daemon-service"
    }
],
```

6. Restart the agent as a daemon service by following these steps:

- a. Run the following command.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
cat /tmp/cwagent-ecs-instance-metric.json \
| sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}| \
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
|xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

- b. Run the following command to launch the daemon service. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
--cluster ${ClusterName} \
--service-name cwagent-daemon-service \
--task-definition ecs-cwagent-daemon-service \
--scheduling-strategy DAEMON \
--region ${Region}
```

If you see this error message, An error occurred (InvalidOperationException) when calling the CreateService operation: Creation of service was not idempotent, you have already created a daemon service named cwagent-daemon-service. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=Region
aws ecs delete-service \
--cluster ${ClusterName} \
--service cwagent-daemon-service \
--region ${Region} \
--force
```

Deploying the AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters

Use the steps in this section to use AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on an Amazon ECS cluster. For more information about the AWS Distro for OpenTelemetry, see [AWS Distro for OpenTelemetry](#).

These steps assume that you already have a cluster running Amazon ECS. This cluster must be deployed with the EC2 launch type. For more information about using AWS Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service for ECS EC2 instance level metrics](#).

Topics

- [Quick setup using AWS CloudFormation \(p. 317\)](#)
- [Manual and custom setup \(p. 318\)](#)

Quick setup using AWS CloudFormation

Download the AWS CloudFormation template file for installing the AWS Distro for OpenTelemetry collector for Amazon ECS on EC2. Run the following curl command.

```
curl -O https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/ecs/aws-otel-ec2-instance-metrics-daemon-deployment-cfn.yaml
```

After you download the template file, open it and replace *PATH_TO_CloudFormation_TEMPLATE* with the path where you saved the template file. Then export the following parameters and run the AWS CloudFormation command, as shown in the following command.

- **Cluster_Name**– The Amazon ECS cluster name
- **AWS_Region**– The Region where the data will be sent
- **PATH_TO_CloudFormation_TEMPLATE**– The path where you saved the AWS CloudFormation template file.
- **command**– To enable the AWS Distro for OpenTelemetry collector to collect the instance-level metrics for Amazon ECS on Amazon EC2, you must specify `--config=/etc/ecs/otel-instance-metrics-config.yaml` for this parameter.

```
ClusterName=Cluster_Name
Region=AWS_Region
command[--config=/etc/ecs/otel-instance-metrics-config.yaml
aws cloudformation create-stack --stack-name AOCECS-${ClusterName}-${Region} \
--template-body file://PATH_TO_CloudFormation_TEMPLATE \
--parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
ParameterKey/CreateIAMRoles,ParameterValue=True \
ParameterKey=command,ParameterValue=${command} \
```

```
--capabilities CAPABILITY_NAMED_IAM \
--region ${Region}
```

After running this command, use the Amazon ECS console to see if the task is running.

Troubleshooting the quick setup

To check the status of the AWS CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack --stack-name AOCECS-$ClusterName-$Region --region $Region
```

If the value of `StackStatus` is anything other than `CREATE_COMPLETE` or `CREATE_IN_PROGRESS`, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name AOCECS-$ClusterName-$Region --region $Region
```

To check the status of the AOCECS daemon service, enter the following command. In the output, you should see that `runningCount` is equal to the `desiredCount` in the deployment section. If it isn't equal, check the failures section in the output.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services AOCECS-daemon-service --cluster $ClusterName --region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the `/aws/ecs/containerinsights/{ClusterName}/performance` log group.

Manual and custom setup

Follow the steps in this section to manually deploy the AWS Distro for OpenTelemetry to collect instance-level metrics from your Amazon ECS clusters that are hosted on Amazon EC2 instances.

Step 1: Necessary roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see [Create IAM policy](#) and [Create IAM role](#).

Step 2: Create the task definition

Create a task definition and use it to launch the AWS Distro for OpenTelemetry as a daemon service.

To use the task definition template to create the task definition, follow the instructions in [Create ECS EC2 Task Definition for EC2 instance with AWS OTel Collector](#).

To use the Amazon ECS console to create the task definition, follow the instructions in [Install AWS OTel Collector by creating Task Definition through AWS console for Amazon ECS EC2 instance metrics](#).

Step 3: Launch the daemon service

To launch the AWS Distro for OpenTelemetry as a daemon service, follow the instructions in [Run your task on the Amazon Elastic Container Service \(Amazon ECS\) using daemon service](#).

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the AWS Distro for OpenTelemetry in your Amazon ECS clusters that are hosted on Amazon EC2 instances. For more information, about creating a configuration file, see [Custom OpenTelemetry Configuration](#). For more information about the options that you can use in the configuration file, see [AWS Container Insights Receiver](#).

Set up Firelens to send logs to CloudWatch Logs

FireLens for Amazon ECS enables you to use task definition parameters to route logs to Amazon CloudWatch Logs for log storage and analytics. FireLens works with [Fluent Bit](#) and [Fluentd](#). We provide an AWS for Fluent Bit image, or you can use your own Fluent Bit or Fluentd image. Creating Amazon ECS task definitions with a FireLens configuration is supported using the AWS SDKs, AWS CLI, and AWS Management Console. For more information about CloudWatch Logs, see [What is CloudWatch Logs?](#).

There are key considerations when using FireLens for Amazon ECS. For more information, see [Considerations](#).

To find the AWS for Fluent Bit images, see [Using the AWS for Fluent Bit image](#).

To create a task definition that uses a FireLens configuration, see [Creating a task definition that uses a FireLens configuration](#).

Example

The following task definition example demonstrates how to specify a log configuration that forwards logs to a CloudWatch Logs log group. For more information, see [What Is Amazon CloudWatch Logs?](#) in the [Amazon CloudWatch Logs User Guide](#).

In the log configuration options, specify the log group name and the Region it exists in. To have Fluent Bit create the log group on your behalf, specify "auto_create_group": "true". You can also specify the task ID as the log stream prefix, which assists in filtering. For more information, see [Fluent Bit Plugin for CloudWatch Logs](#).

```
{  
  "family": "firelens-example-cloudwatch",  
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",  
  "containerDefinitions": [  
    {  
      "essential": true,  
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",  
      "name": "log_router",  
      "firelensConfiguration": {  
        "type": "fluentbit"  
      },  
      "logConfiguration": {  
        "logDriver": "awslogs",  
        "options": {  
          "awslogs-group": "firelens-container",  
          "awslogs-region": "us-west-2",  
          "awslogs-create-group": "true",  
          "awslogs-stream-prefix": "firelens"  
        }  
      },  
      "memoryReservation": 50  
    },  
    {  
      "essential": true,  
      "image": "nginx",  
      "name": "app",  
    }  
  ]  
}
```

```
    "logConfiguration": {  
        "logDriver": "awsfirelens",  
        "options": {  
            "Name": "cloudwatch",  
            "region": "us-west-2",  
            "log_key": "log",  
            "log_group_name": "/aws/ecs/containerinsights/  
$(ecs_cluster)/application",  
            "auto_create_group": "true",  
            "log_stream_name": "$(ecs_task_id)"  
        }  
    },  
    "memoryReservation": 100  
}  
]  
}
```

Setting up Container Insights on Amazon EKS and Kubernetes

The overall process for setting up Container Insights on Amazon EKS or Kubernetes is as follows:

1. Verify that you have the necessary prerequisites.
2. Set up the CloudWatch agent or the AWS Distro for OpenTelemetry as a DaemonSet on your cluster to send metrics to CloudWatch.

Set up Fluent Bit or FluentD to send logs to CloudWatch Logs.

You can perform these steps at once as part of the quick start setup if you are using the CloudWatch agent, or do them separately.
3. (Optional) Set up Amazon EKS control plane logging.
4. (Optional) Set up the CloudWatch agent as a StatsD endpoint on the cluster to send StatsD metrics to CloudWatch.
5. (Optional) Enable App Mesh Envoy Access Logs.

Topics

- [Verify prerequisites \(p. 320\)](#)
- [Using the CloudWatch agent \(p. 322\)](#)
- [Using AWS Distro for OpenTelemetry \(p. 327\)](#)
- [Send logs to CloudWatch Logs \(p. 328\)](#)
- [Updating or deleting Container Insights on Amazon EKS and Kubernetes \(p. 342\)](#)

Verify prerequisites

Before you install Container Insights on Amazon EKS or Kubernetes, verify the following. These prerequisites apply whether you are using the CloudWatch agent or AWS Distro for OpenTelemetry to set up Container Insights on Amazon EKS clusters.

- You have a functional Amazon EKS or Kubernetes cluster with nodes attached in one of the Regions that supports the Container Insights for Amazon EKS and Kubernetes. For the list of supported Regions, see [Using Container Insights \(p. 305\)](#).
- You have `kubectl` installed and running. For more information, see [Installing `kubectl`](#) in the *Amazon EKS User Guide*.

- If you're using Kubernetes running on AWS instead of using Amazon EKS, the following prerequisites are also necessary:
 - Be sure that your Kubernetes cluster has enabled role-based access control (RBAC). For more information, see [Using RBAC Authorization](#) in the Kubernetes Reference.
 - Your kubelet has enabled Webhook authorization mode. For more information, see [Kubelet authentication/authorization](#) in the Kubernetes Reference.
 - Your container runtime is Docker.

You must also grant IAM permissions to enable your Amazon EKS worker nodes to send metrics and logs to CloudWatch. There are two ways to do this:

- Attach a policy to the IAM role of your worker nodes. This works for both Amazon EKS clusters and other Kubernetes clusters.
- Use an IAM role for service accounts for the cluster, and attach the policy to this role. This works only for Amazon EKS clusters.

The first option grants permissions to CloudWatch for the entire node, while using an IAM role for the service account gives CloudWatch access to only the appropriate daemonset pods.

Attaching a policy to the IAM role of your worker nodes

Follow these steps to attach the policy to the IAM role of your worker nodes. This works for both Amazon EKS clusters and Kubernetes clusters outside of Amazon EKS.

To attach the necessary policy to the IAM role for your worker nodes

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Select one of the worker node instances and choose the IAM role in the description.
3. On the IAM role page, choose **Attach policies**.
4. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find this policy.
5. Choose **Attach policies**.

If you're running a Kubernetes cluster outside Amazon EKS, you might not already have an IAM role attached to your worker nodes. If not, you must first attach an IAM role to the instance and then add the policy as explained in the previous steps. For more information on attaching a role to an instance, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide for Windows Instances*.

If you're running a Kubernetes cluster outside Amazon EKS and you want to collect EBS volume IDs in the metrics, you must add another policy to the IAM role attached to the instance. Add the following as an inline policy. For more information, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "ec2:DescribeVolumes"  
            ],  
            "Resource": "*",  
            "Effect": "Allow"  
        }  
    ]  
}
```

}

Using an IAM service account role

This method works only on Amazon EKS clusters.

To grant permission to CloudWatch using an IAM service account role

1. If you haven't already, enable IAM roles for service accounts on your cluster. For more information, see [Enabling IAM roles for service accounts on your cluster](#).
2. If you haven't already, create the IAM role for your service account. For more information, see [Creating an IAM role and policy for your service account](#).

When you create the role, attach the **CloudWatchAgentServerPolicy** IAM policy to the role in addition to the policy that you create for the role.

3. If you haven't already, associate the IAM role with a service account in your cluster. For more information, see [Specifying an IAM role for your service account](#)

Using the CloudWatch agent

Use the instructions in these sections to set up Container Insights on an Amazon EKS cluster by using the CloudWatch agent.

Topics

- [Quick Start setup for Container Insights on Amazon EKS and Kubernetes \(p. 322\)](#)
- [Set up the CloudWatch agent to collect cluster metrics \(p. 324\)](#)

Quick Start setup for Container Insights on Amazon EKS and Kubernetes

To complete the setup of Container Insights, you can follow the quick start instructions in this section.

Important

Before completing the steps in this section, you must have verified the prerequisites including IAM permissions. For more information, see [Verify prerequisites \(p. 320\)](#).

Alternatively, you can instead follow the instructions in the following two sections, [Set up the CloudWatch agent to collect cluster metrics \(p. 324\)](#) and [Send logs to CloudWatch Logs \(p. 328\)](#). Those sections provide more configuration details on how the CloudWatch agent works with Amazon EKS and Kubernetes, but require you to perform more installation steps.

Note

Amazon has now launched Fluent Bit as the default log solution for Container Insights with significant performance gains. We recommend that you use Fluent Bit instead of Fluentd.

Quick Start with the CloudWatch agent and Fluent Bit

There are two configurations for Fluent Bit: an optimized version and a version that provides an experience more similar to FluentD. The Quick Start configuration uses the optimized version. For more details about the FluentD-compatible configuration, see [Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs \(p. 329\)](#).

To deploy Container Insights using the quick start, enter the following command.

Note

The following setup step pulls the container image from Docker Hub as an anonymous user by default. This pull may be subject to a rate limit. For more information, see [CloudWatch agent container image \(p. 306\)](#).

```
ClusterName=<my-cluster-name>
RegionName=<my-cluster-region>
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off' ||
FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' || FluentBitHttpServer='On'
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-
mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluent-
bit-quickstart.yaml | sed 's/{{cluster_name}}/${ClusterName}';s/
{{region_name}}/${RegionName}';s/{{http_server_toggle}}/'${FluentBitHttpServer}'';s/
{{http_server_port}}/'${FluentBitHttpPort}'';s/
{{read_from_head}}/'${FluentBitReadFromHead}'';s/
{{read_from_tail}}/'${FluentBitReadFromTail}'' | kubectl apply -f -
```

In this command, *my-cluster-name* is the name of your Amazon EKS or Kubernetes cluster, and *my-cluster-region* is the name of the Region where the logs are published. We recommend that you use the same Region where your cluster is deployed to reduce the AWS outbound data transfer costs.

For example, to deploy Container Insights on the cluster named MyCluster and publish the logs and metrics to US West (Oregon), enter the following command.

```
ClusterName='MyCluster'
LogRegion='us-west-2'
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off' ||
FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' || FluentBitHttpServer='On'
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-
mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluent-
bit-quickstart.yaml | sed 's/{{cluster_name}}/${ClusterName}';s/
{{region_name}}/${LogRegion}';s/{{http_server_toggle}}/'${FluentBitHttpServer}'';s/
{{http_server_port}}/'${FluentBitHttpPort}'';s/
{{read_from_head}}/'${FluentBitReadFromHead}'';s/
{{read_from_tail}}/'${FluentBitReadFromTail}'' | kubectl apply -f -
```

Migrating from Fluentd

If you already have Fluentd configured and want to move to Fluent Bit, you must delete the FluentD pods after you install Fluent Bit. You can use the following command to delete FluentD.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/
latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-
monitoring/fluentd/fluentd.yaml | kubectl delete -f -
kubectl delete configmap cluster-info -n amazon-cloudwatch
```

Deleting Container Insights

If you want to remove Container Insights after using the quick start setup, enter the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-
mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluent-
bit-quickstart.yaml | sed 's/{{cluster_name}}/${ClusterName}';s/
{{region_name}}/${LogRegion}';s/{{http_server_toggle}}/'${FluentBitHttpServer}'';s/
{{http_server_port}}/'${FluentBitHttpPort}'';s/
```

```
 {{read_from_head}}"/'`${FluentBitReadFromHead}'"/;s/  
 {{read_from_tail}}"/'`${FluentBitReadFromTail}'`/ | kubectl delete -f -
```

Quick Start with the CloudWatch agent and Fluentd

If you are already using Fluentd in your Kubernetes cluster and want to extend it to be the log solution for Container Insights, we provide a FluentD configuration for you to do so.

To deploy the CloudWatch agent and Fluentd using the quick start, use the following command. The following setup contains a community supported FluentD container image. You can replace the image with your own FluentD image as long as it meets the FluentD image requirements. For more information, see [\(Optional\) Set up FluentD as a DaemonSet to send logs to CloudWatch Logs \(p. 335\)](#).

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluentd-quickstart.yaml | sed "s/{{cluster_name}}/cluster-name/;s/{{region_name}}/cluster-region/" | kubectl apply -f -
```

In this command, `cluster-name` is the name of your Amazon EKS or Kubernetes cluster, and `cluster-region` is the name of the Region where the logs are published. We recommend that you use the same Region where your cluster is deployed to reduce the AWS outbound data transfer costs.

For example, to deploy Container Insights on the cluster named `MyCluster` and publish the logs and metrics to US West (Oregon), enter the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluentd-quickstart.yaml | sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/us-west-2/" | kubectl apply -f -
```

Deleting Container Insights

If you want to remove Container Insights after using the quick start setup, enter the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluentd-quickstart.yaml | sed "s/{{cluster_name}}/cluster-name/;s/{{region_name}}/cluster-region/" | kubectl delete -f -
```

Set up the CloudWatch agent to collect cluster metrics

To set up Container Insights to collect metrics, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes \(p. 322\)](#) or you can follow the steps in this section. In the following steps, you set up the CloudWatch agent to be able to collect metrics from your clusters.

Step 1: Create a namespace for CloudWatch

Use the following step to create a Kubernetes namespace called `amazon-cloudwatch` for CloudWatch. You can skip this step if you have already created this namespace.

To create a namespace for CloudWatch

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

Step 2: Create a service account in the cluster

Use the following step to create a service account for the CloudWatch agent, if you do not already have one.

To create a service account for the CloudWatch agent

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent-serviceaccount.yaml
```

If you didn't follow the previous steps, but you already have a service account for the CloudWatch agent that you want to use, you must ensure that it has the following rules. Additionally, in the rest of the steps in the Container Insights installation, you must use the name of that service account instead of `cloudwatch-agent`.

```
rules:  
- apiGroups: [""]  
  resources: ["pods", "nodes", "endpoints"]  
  verbs: ["watch", "list"]  
- apiGroups: [""]  
  resources: ["nodes/proxy"]  
  verbs: ["get"]  
- apiGroups: [""]  
  resources: ["nodes/stats", "configmaps", "events"]  
  verbs: ["create"]  
- apiGroups: [""]  
  resources: ["configmaps"]  
  resourceNames: ["cwagent-clusterleader"]  
  verbs: ["get", "update"]  
- nonResourceURLs: ["/metrics"]  
  verbs: ["get"]
```

Step 3: Create a ConfigMap for the CloudWatch agent

Use the following steps to create a ConfigMap for the CloudWatch agent.

To create a ConfigMap for the CloudWatch agent

- Download the ConfigMap YAML to your `kubectl` client host by running the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent-configmap.yaml
```

- Edit the downloaded YAML file, as follows:

- cluster_name** – In the `kubernetes` section, replace `{{cluster-name}}` with the name of your cluster. Remove the `{}{}` characters. Alternatively, if you're using an Amazon EKS cluster, you can delete the `"cluster_name"` field and value. If you do, the CloudWatch agent detects the cluster name from the Amazon EC2 tags.
- (Optional) Make further changes to the ConfigMap based on your monitoring requirements, as follows:
 - metrics_collection_interval** – In the `kubernetes` section, you can specify how often the agent collects metrics. The default is 60 seconds. The default cadvisor collection interval in kubelet is 15 seconds, so don't set this value to less than 15 seconds.

- **endpoint_override** – In the logs section, you can specify the CloudWatch Logs endpoint if you want to override the default endpoint. You might want to do this if you're publishing from a cluster in a VPC and you want the data to go to a VPC endpoint.
- **force_flush_interval** – In the logs section, you can specify the interval for batching log events before they are published to CloudWatch Logs. The default is 5 seconds.
- **region** – By default, the agent published metrics to the Region where the worker node is located. To override this, you can add a `region` field in the `agent` section: for example, `"region": "us-west-2"`.
- **statsd** section – If you want the CloudWatch Logs agent to also run as a StatsD listener in each worker node of your cluster, you can add a `statsd` section to the `metrics` section, as in the following example. For information about other StatsD options for this section, see [Retrieve custom metrics with StatsD \(p. 562\)](#).

```
"metrics": {  
    "metrics_collected": {  
        "statsd": {  
            "service_address": ":8125"  
        }  
    }  
}
```

A full example of the JSON section is as follows.

```
{  
    "agent": {  
        "region": "us-east-1"  
    },  
    "logs": {  
        "metrics_collected": {  
            "kubernetes": {  
                "cluster_name": "MyCluster",  
                "metrics_collection_interval": 60  
            }  
        },  
        "force_flush_interval": 5,  
        "endpoint_override": "logs.us-east-1.amazonaws.com"  
    },  
    "metrics": {  
        "metrics_collected": {  
            "statsd": {  
                "service_address": ":8125"  
            }  
        }  
    }  
}
```

4. Create the ConfigMap in the cluster by running the following command.

```
kubectl apply -f cwagent-configmap.yaml
```

Step 4: Deploy the CloudWatch agent as a DaemonSet

To finish the installation of the CloudWatch agent and begin collecting container metrics, use the following steps.

To deploy the CloudWatch agent as a DaemonSet

1. • If you do not want to use StatsD on the cluster, enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

- If you do want to use StatsD, follow these steps:
 - a. Download the DaemonSet YAML to your `kubectl` client host by running the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

- b. Uncomment the port section in the `cwagent-daemonset.yaml` file as in the following:

```
ports:  
  - containerPort: 8125  
    hostPort: 8125  
    protocol: UDP
```

- c. Deploy the CloudWatch agent in your cluster by running the following command.

```
kubectl apply -f cwagent-daemonset.yaml
```

2. Validate that the agent is deployed by running the following command.

```
kubectl get pods -n amazon-cloudwatch
```

When complete, the CloudWatch agent creates a log group named `/aws/containerinsights/Cluster Name/performance` and sends the performance log events to this log group. If you also set up the agent as a StatsD listener, the agent also listens for StatsD metrics on port 8125 with the IP address of the node where the application pod is scheduled.

Troubleshooting

If the agent doesn't deploy correctly, try the following:

- Run the following command to get the list of pods.

```
kubectl get pods -n amazon-cloudwatch
```

- Run the following command and check the events at the bottom of the output.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- Run the following command to check the logs.

```
kubectl logs pod-name -n amazon-cloudwatch
```

Using AWS Distro for OpenTelemetry

You can set up Container Insights to collect metrics from Amazon EKS clusters by using the AWS Distro for OpenTelemetry collector. For more information about the AWS Distro for OpenTelemetry, see [AWS Distro for OpenTelemetry](#).

How you set up Container Insights depends on whether the cluster is hosted on Amazon EC2 instances or on AWS Fargate (Fargate).

Amazon EKS clusters hosted on Amazon EC2

If you have not already done so, make sure that you have fulfilled the prerequisites including the necessary IAM roles. For more information, see [Verify prerequisites \(p. 320\)](#).

Amazon provides a Helm chart that you can use to set up the monitoring of Amazon Elastic Kubernetes Service on Amazon EC2. This monitoring uses the AWS Distro for OpenTelemetry(ADOT) Collector for metrics and Fluent Bit for logs. Therefore, the Helm chart is useful for customers who use Amazon EKS on Amazon EC2 and want to collect metrics and logs to send to CloudWatch Container Insights. For more information about this Helm chart, see [ADOT Helm chart for EKS on EC2 metrics and logs to Amazon CloudWatch Container Insights](#).

Alternatively, you can also use the instructions in the rest of this section.

First, deploy the AWS Distro for OpenTelemetry collector as a DaemonSet by entering the following command.

```
curl https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/deployment-template/eks/otel-container-insights-infra.yaml | kubectl apply -f -
```

To confirm that the collector is running, enter the following command.

```
kubectl get pods -l name=aws-otel-eks-ci -n aws-otel-eks
```

If the output of this command includes multiple pods in the `Running` state, the collector is running and collecting metrics from the cluster. The collector creates a log group named `aws/containerinsights/<cluster-name>/performance` and sends the performance log events to it.

For information about how to see your Container Insights metrics in CloudWatch, see [Viewing Container Insights metrics \(p. 344\)](#).

AWS has also provided documentation on GitHub for this scenario. If you want to customize the metrics and logs published by Container Insights, see <https://aws-otel.github.io/docs/getting-started/container-insights/eks-infra>.

Amazon EKS clusters hosted on Fargate

For instructions for how to configure and deploy an ADOT Collector to collect system metrics from workloads deployed to an Amazon EKS cluster on Fargate and send them to CloudWatch Container Insights, see [Container Insights EKS Fargate](#) in the AWS Distro for OpenTelemetry documentation.

Send logs to CloudWatch Logs

To send logs from your containers to Amazon CloudWatch Logs, you can use Fluent Bit or Fluentd. For more information, see [Fluent Bit](#) and [Fluentd](#).

If you are not already using Fluentd, we recommend that you use Fluent Bit for the following reasons:

- Fluent Bit has a smaller resource footprint and is more resource-efficient with memory and CPU usage than FluentD. For a more detailed comparison, see [Fluent Bit and Fluentd performance comparison \(p. 329\)](#).

- The Fluent Bit image is developed and maintained by AWS. This gives AWS the ability to adopt new Fluent Bit image features and respond to issues much quicker.

Topics

- [Fluent Bit and Fluentd performance comparison \(p. 329\)](#)
- [Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs \(p. 329\)](#)
- [\(Optional\) Set up FluentD as a DaemonSet to send logs to CloudWatch Logs \(p. 335\)](#)
- [\(Optional\) Set up Amazon EKS control plane logging \(p. 339\)](#)
- [\(Optional\) Enable App Mesh Envoy access logs \(p. 339\)](#)
- [\(Optional\) Enable the Use_Kubelet feature for large clusters \(p. 340\)](#)

Fluent Bit and Fluentd performance comparison

The following tables show the the performance advantage that Fluent Bit has over FluentD in memory and CPU usages. The following numbers are just for reference and might change depending on the environment.

| Logs per second | Fluentd CPU usage | Fluent Bit CPU usage with Fluentd-compatible configuration | Fluent Bit CPU usage with optimized configuration |
|-----------------|-------------------|--|---|
| 100 | 0.35 vCPU | 0.02 vCPU | 0.02 vCPU |
| 1,000 | 0.32 vCPU | 0.14 vCPU | 0.11 vCPU |
| 5,000 | 0.85 vCPU | 0.48 vCPU | 0.30 vCPU |
| 10,000 | 0.94 vCPU | 0.60 vCPU | 0.39 vCPU |

| Logs per second | Fluentd memory usage | Fluent Bit memory usage with Fluentd-compatible configuration | Fluent Bit memory usage with optimized configuration |
|-----------------|----------------------|---|--|
| 100 | 153 MB | 46 MB | 37 MB |
| 1,000 | 270 MB | 45 MB | 40 MB |
| 5,000 | 320 MB | 55 MB | 45 MB |
| 10,000 | 375 MB | 92 MB | 75 MB |

Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs

The following sections help you deploy Fluent Bit to send logs from containers to CloudWatch Logs.

Topics

- [Differences if you're already using Fluentd \(p. 330\)](#)
- [Setting up Fluent Bit \(p. 330\)](#)
- [Multiline log support \(p. 332\)](#)

- [\(Optional\) Reducing the log volume from Fluent Bit \(p. 333\)](#)
- [Troubleshooting \(p. 334\)](#)
- [Dashboard \(p. 334\)](#)

Differences if you're already using Fluentd

If you are already using Fluentd to send logs from containers to CloudWatch Logs, read this section to see the differences between Fluentd and Fluent Bit. If you are not already using Fluentd with Container Insights, you can skip to [Setting up Fluent Bit \(p. 330\)](#).

We provide two default configurations for Fluent Bit:

- **Fluent Bit optimized configuration** — A configuration aligned with Fluent Bit best practices.
- **Fluentd-compatible configuration** — A configuration that is aligned with Fluentd behavior as much as possible.

The following list explains the differences between Fluentd and each Fluent Bit configuration in detail.

- **Differences in log stream names** — If you use the Fluent Bit optimized configuration, the log stream names will be different.

Under /aws/containerinsights/Cluster_Name/application

- Fluent Bit optimized configuration sends logs to `kubernetes-nodeName-application.var.log.containers.kubernetes-podName_kubernetes-namespace_kubernetes-container-name-kubernetes-containerID`
- Fluentd sends logs to `kubernetes-podName_kubernetes-namespace_kubernetes-containerName_kubernetes-containerID`

Under /aws/containerinsights/Cluster_Name/host

- Fluent Bit optimized configuration sends logs to `kubernetes-nodeName.host-log-file`
- Fluentd sends logs to `host-log-file-Kubernetes-NodePrivateIp`

Under /aws/containerinsights/Cluster_Name/dataplane

- Fluent Bit optimized configuration sends logs to `kubernetes-nodeName.dataplaneServiceLog`
- Fluentd sends logs to `dataplaneServiceLog-Kubernetes-nodeName`
- The kube-proxy and aws-node log files that Container Insights writes are in different locations. In FluentD configuration, they are in /aws/containerinsights/Cluster_Name/application. In the Fluent Bit optimized configuration, they are in /aws/containerinsights/Cluster_Name/dataplane.
- Most metadata such as pod_name and namespace_name are the same in Fluent Bit and Fluentd, but the following are different.
 - The Fluent Bit optimized configuration uses docker_id and Fluentd use Docker.container_id.
 - Both Fluent Bit configurations do not use the following metadata. They are present only in Fluentd: container_image_id, master_url, namespace_id, and namespace_labels.

Setting up Fluent Bit

To set up Fluent Bit to collect logs from your containers, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes \(p. 322\)](#) or you can follow the steps in this section.

With either method, the IAM role that is attached to the cluster nodes must have sufficient permissions. For more information about the permissions required to run an Amazon EKS cluster, see [Amazon EKS IAM Policies, Roles, and Permissions](#) in the *Amazon EKS User Guide*.

In the following steps, you set up Fluent Bit as a daemonSet to send logs to CloudWatch Logs. When you complete this step, Fluent Bit creates the following log groups if they don't already exist.

| Log group name | Log source |
|--|--|
| /aws/containerinsights/ <i>Cluster_Name</i> /application | All log files in /var/log/containers |
| /aws/containerinsights/ <i>Cluster_Name</i> /host | Logs from /var/log/dmesg, /var/log/secure, and /var/log/messages |
| /aws/containerinsights/ <i>Cluster_Name</i> /dataplane | The logs in /var/log/journal for kubelet.service, kubeproxy.service, and docker.service. |

To install Fluent Bit to send logs from containers to CloudWatch Logs

- If you don't already have a namespace called `amazon-cloudwatch`, create one by entering the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

- Run the following command to create a ConfigMap named `cluster-info` with the cluster name and the Region to send logs to. Replace `cluster-name` and `cluster-region` with your cluster's name and Region.

```
ClusterName=cluster-name
RegionName=cluster-region
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off'|||
FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' ||| FluentBitHttpServer='On'
kubectl create configmap fluent-bit-cluster-info \
--from-literal=cluster.name=${ClusterName} \
--from-literal=http.server=${FluentBitHttpServer} \
--from-literal=http.port=${FluentBitHttpPort} \
--from-literal=read.head=${FluentBitReadFromHead} \
--from-literal=read.tail=${FluentBitReadFromTail} \
--from-literal=logs.region=${RegionName} -n amazon-cloudwatch
```

In this command, the `FluentBitHttpServer` for monitoring plugin metrics is on by default. To turn it off, change the third line in the command to `FluentBitHttpServer=''` (empty string) in the command.

Also by default, Fluent Bit reads log files from the tail, and will capture only new logs after it is deployed. If you want the opposite, set `FluentBitReadFromHead='On'` and it will collect all logs in the file system.

- Download and deploy the Fluent Bit daemonset to the cluster by running one of the following commands.
 - If you want the Fluent Bit optimized configuration, run this command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit.yaml
```

- If you want the Fluent Bit configuration that is more similar to Fluentd, run this command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-compatible.yaml
```

4. Validate the deployment by entering the following command. Each node should have one pod named **fluent-bit-***.

```
kubectl get pods -n amazon-cloudwatch
```

The above steps create the following resources in the cluster:

- A service account named **Fluent-Bit** in the `amazon-cloudwatch` namespace. This service account is used to run the Fluent Bit daemonSet. For more information, see [Managing Service Accounts](#) in the Kubernetes Reference.
- A cluster role named **Fluent-Bit-role** in the `amazon-cloudwatch` namespace. This cluster role grants get, list, and watch permissions on pod logs to the **Fluent-Bit** service account. For more information, see [API Overview](#) in the Kubernetes Reference.
- A ConfigMap named **Fluent-Bit-config** in the `amazon-cloudwatch` namespace. This ConfigMap contains the configuration to be used by Fluent Bit. For more information, see [Configure a Pod to Use a ConfigMap](#) in the Kubernetes Tasks documentation.

If you want to verify your Fluent Bit setup, follow these steps.

Verify the Fluent Bit setup

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Make sure that you're in the Region where you deployed Fluent Bit.
4. Check the list of log groups in the Region. You should see the following:
 - `/aws/containerinsights/Cluster_Name/application`
 - `/aws/containerinsights/Cluster_Name/host`
 - `/aws/containerinsights/Cluster_Name/dataplane`
5. Navigate to one of these log groups and check the **Last Event Time** for the log streams. If it is recent relative to when you deployed Fluent Bit, the setup is verified.

There might be a slight delay in creating the `/dataplane` log group. This is normal as these log groups only get created when Fluent Bit starts sending logs for that log group.

Multiline log support

By default, the multiline log entry starter is any character with no white space. This means that all log lines that start with a character that does not have white space are considered as a new multiline log entry.

If your own application logs use a different multiline starter, you can support them by making two changes in the `Fluent-Bit.yaml` file.

First, exclude them from the default input by adding the pathnames of your log files to an `exclude_path` field in the `containers` section of `Fluent-Bit.yaml`. The following is an example.

```
[INPUT]
```

| | |
|--------------|--|
| Name | tail |
| Tag | application.* |
| Exclude_Path | <i>full.pathname.of.log_file*</i> , <i>full.pathname.of.log_file2*</i> |
| Path | /var/log/containers/*.log |

Next, add a block for your log files to the `Fluent-Bit.yaml` file. Refer to the cloudwatch-agent log configuration example below which uses a timestamp regular expression as the multiline starter.

```
application-log.conf: |
  [INPUT]
    Name          tail
    Tag           application.*
    Path          /var/log/containers/cloudwatch-agent*
    Docker_Mode   On
    Docker_Mode_Flush 5
    Docker_Mode_Parser cwagent_firstline
    Parser        docker
    DB            /fluent-bit/state/flb_cwagent.db
    Mem_Buf_Limit 5MB
    Skip_Long_Lines On
    Refresh_Interval 10

parsers.conf: |
  [PARSER]
    Name          cwagent_firstline
    Format        regex
    Regex         (?<log>(?<="log":")\d{4}[\/-]\d{1,2}[\/-]\d{1,2}[ \T]\d{2}:
    \d{2}:\d{2}(?!\.)*?)(?<!\\").*(?<stream>(?<="stream":").*?).*?(?<time>\d{4}-\d{1,2}-
    \d{1,2}T\d{2}:\d{2}:\d{2}\.\w*).*(?=)
    Time_Key     time
    Time_Format  %Y-%m-%dT%H:%M:%S.%LZ
```

(Optional) Reducing the log volume from Fluent Bit

By default, we send Fluent Bit application logs and Kubernetes metadata to CloudWatch. If you want to reduce the volume of data being sent to CloudWatch, you can stop one or both of these data sources from being sent to CloudWatch.

To stop Fluent Bit application logs, remove the following section from the `Fluent-Bit.yaml` file.

```
[INPUT]
  Name          tail
  Tag           application.*
  Path          /var/log/containers/fluent-bit*
  Parser        docker
  DB            /fluent-bit/state/flb_log.db
  Mem_Buf_Limit 5MB
  Skip_Long_Lines On
  Refresh_Interval 10
```

To remove Kubernetes metadata from being appended to log events that are sent to CloudWatch, add the following filters to the `application-log.conf` section in the `Fluent-Bit.yaml` file.

```
application-log.conf: |
  [FILTER]
    Name          nest
    Match         application.*
    Operation    lift
    Nested_under kubernetes
    Add_prefix   Kube.
```

```
[FILTER]
  Name          modify
  Match         application.*
  Remove        Kube.<Metadata_1>
  Remove        Kube.<Metadata_2>
  Remove        Kube.<Metadata_3>

[FILTER]
  Name          nest
  Match         application.*
  Operation    nest
  Wildcard     Kube./*
  Nested_under kubernetes
  Remove_prefix Kube.
```

Troubleshooting

If you don't see these log groups and are looking in the correct Region, check the logs for the Fluent Bit daemonSet pods to look for the error.

Run the following command and make sure that the status is `Running`.

```
kubectl get pods -n amazon-cloudwatch
```

If the logs have errors related to IAM permissions, check the IAM role that is attached to the cluster nodes. For more information about the permissions required to run an Amazon EKS cluster, see [Amazon EKS IAM Policies, Roles, and Permissions](#) in the *Amazon EKS User Guide*.

If the pod status is `CreateContainerConfigError`, get the exact error by running the following command.

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

Dashboard

You can create a dashboard to monitor metrics of each running plugin. You can see data for input and output bytes and for record processing rates as well as output errors and retry/failed rates. To view these metrics, you will need to install the CloudWatch agent with Prometheus metrics collection for Amazon EKS and Kubernetes clusters. For more information about how to set up the dashboard, see [Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters \(p. 414\)](#).

Note

Before you can set up this dashboard, you must set up Container Insights for Prometheus metrics. For more information, see [Container Insights Prometheus metrics monitoring \(p. 374\)](#).

To create a dashboard for the Fluent Bit Prometheus metrics

1. Create environment variables, replacing the values on the right in the following lines to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-west-1
CLUSTER_NAME=your_kubernetes_cluster_name
```

2. Create the dashboard by running the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_cloudwatch_dashboards/fluent-bit/cw_dashboard_fluent_bit.json \
```

```
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \  
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \  
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --dashboard-body
```

(Optional) Set up FluentD as a DaemonSet to send logs to CloudWatch Logs

Warning

Container Insights Support for FluentD is now in maintenance mode, which means that AWS will not provide any further updates for FluentD and that we are planning to deprecate it in near future. Additionally, the current FluentD configuration for Container Insights is using an old version of the FluentD Image fluent/fluentd-kubernetes-daemonset:v1.7.3-debian-cloudwatch-1.0 which does not have the latest improvement and security patches. For the latest FluentD image supported by the open source community, see [fluentd-kubernetes-daemonset](#).

We strongly recommend that you migrate to use FluentBit with Container Insights whenever possible. Using FluentBit as the log forwarder for Container Insights provides significant performance gains.

For more information, see [Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs \(p. 329\)](#) and [Differences if you're already using Fluentd \(p. 330\)](#).

To set up FluentD to collect logs from your containers, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes \(p. 322\)](#) or you can follow the steps in this section. In the following steps, you set up FluentD as a DaemonSet to send logs to CloudWatch Logs. When you complete this step, FluentD creates the following log groups if they don't already exist.

| Log group name | Log source |
|--|--|
| /aws/containerinsights/ <i>Cluster_Name</i> /application | All log files in /var/log/containers |
| /aws/containerinsights/ <i>Cluster_Name</i> /host | Logs from /var/log/dmesg, /var/log/secure, and /var/log/messages |
| /aws/containerinsights/ <i>Cluster_Name</i> /dataplane | The logs in /var/log/journal for kubelet.service, kubeproxy.service, and docker.service. |

Step 1: Create a namespace for CloudWatch

Use the following step to create a Kubernetes namespace called `amazon-cloudwatch` for CloudWatch. You can skip this step if you have already created this namespace.

To create a namespace for CloudWatch

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

Step 2: Install FluentD

Start this process by downloading FluentD. When you finish these steps, the deployment creates the following resources on the cluster:

- A service account named `fluentd` in the `amazon-cloudwatch` namespace. This service account is used to run the FluentD DaemonSet. For more information, see [Managing Service Accounts](#) in the Kubernetes Reference.
- A cluster role named `fluentd` in the `amazon-cloudwatch` namespace. This cluster role grants `get`, `list`, and `watch` permissions on pod logs to the `fluentd` service account. For more information, see [API Overview](#) in the Kubernetes Reference.
- A ConfigMap named `fluentd-config` in the `amazon-cloudwatch` namespace. This ConfigMap contains the configuration to be used by FluentD. For more information, see [Configure a Pod to Use a ConfigMap](#) in the Kubernetes Tasks documentation.

To install FluentD

1. Create a ConfigMap named `cluster-info` with the cluster name and the AWS Region that the logs will be sent to. Run the following command, updating the placeholders with your cluster and Region names.

```
kubectl create configmap cluster-info \
--from-literal=cluster.name=cluster_name \
--from-literal=logs.region=region_name -n amazon-cloudwatch
```

2. Download and deploy the FluentD DaemonSet to the cluster by running the following command. Make sure that you are using the container image with correct architecture. The example manifest only works on x86 instances and will enter `CrashLoopBackOff` if you have Advanced RISC Machine (ARM) instances in your cluster. The FluentD daemonSet does not have an official multi-architecture Docker image that enables you to use one tag for multiple underlying images and let the container runtime pull the right one. The FluentD ARM image uses a different tag with an `arm64` suffix.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/
container-insights-monitoring/fluentd/fluentd.yaml
```

Note

Because of a recent change to optimize the FluentD configuration and minimize the impact of FluentD API requests on Kubernetes API endpoints, the "Watch" option for Kubernetes filters has been disabled by default. For more details, see [fluent-plugin-kubernetes_metadata_filter](#).

3. Validate the deployment by running the following command. Each node should have one pod named `fluentd-cloudwatch-*`.

```
kubectl get pods -n amazon-cloudwatch
```

Step 3: Verify the FluentD setup

To verify your FluentD setup, use the following steps.

To verify the FluentD setup for Container Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**. Make sure that you're in the Region where you deployed FluentD to your containers.

In the list of log groups in the Region, you should see the following:

- `/aws/containerinsights/Cluster_Name/application`

- /aws/containerinsights/*Cluster_Name*/host
- /aws/containerinsights/*Cluster_Name*/dataplane

If you see these log groups, the FluentD setup is verified.

Multiline log support

On August 19 2019, we added multiline log support for the logs collected by FluentD.

By default, the multiline log entry starter is any character with no white space. This means that all log lines that start with a character that does not have white space are considered as a new multiline log entry.

If your own application logs use a different multiline starter, you can support them by making two changes in the `fluentd.yaml` file.

First, exclude them from the default multiline support by adding the pathnames of your log files to an `exclude_path` field in the `containers` section of `fluentd.yaml`. The following is an example.

```
<source>
  @type tail
  @id in_tail_container_logs
  @label @containers
  path /var/log/containers/*.log
  exclude_path [ "full.pathname.of.log_file*", "full.pathname.of.log_file2*" ]
```

Next, add a block for your log files to the `fluentd.yaml` file. The example below is used for the CloudWatch agent's log file, which uses a timestamp regular expression as the multiline starter. You can copy this block and add it to `fluentd.yaml`. Change the indicated lines to reflect your application log file name and the multiline starter that you want to use.

```
<source>
  @type tail
  @id in_tail_cwagent_logs
  @label @cwagentlogs
  path /var/log/containers/cloudwatch-agent*
  pos_file /var/log/cloudwatch-agent.log.pos
  tag *
  read_from_head true
  <parse>
    @type json
    time_format %Y-%m-%dT%H:%M:%S.%NZ
  </parse>
</source>
```

```
<label @cwagentlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_cwagent
  </filter>

  <filter **>
    @type record_transformer
    @id filter_cwagent_stream_transformer
```

```
<record>
    stream_name ${tag_parts[3]}
</record>
</filter>

<filter **>
    @type concat
    key log
    multiline_start_regexp /^d{4}[-]\d{1,2}[-]\d{1,2}/
    separator ""
    flush_interval 5
    timeout_label @NORMAL
</filter>

<match **>
    @type relabel
    @label @NORMAL
</match>
</label>
```

(Optional) Reducing the log volume from FluentD

By default, we send FluentD application logs and Kubernetes metadata to CloudWatch. If you want to reduce the volume of data being sent to CloudWatch, you can stop one or both of these data sources from being sent to CloudWatch.

To stop FluentD application logs, remove the following section from the fluentd.yaml file.

```
<source>
    @type tail
    @id in_tail_fluentd_logs
    @label @fluentdlogs
    path /var/log/containers/fluentd*
    pos_file /var/log/fluentd.log.pos
    tag *
    read_from_head true
    <parse>
        @type json
        time_format %Y-%m-%dT%H:%M:%S.%NZ
    </parse>
</source>

<label @fluentdlogs>
    <filter **>
        @type kubernetes_metadata
        @id filter_kube_metadata_fluentd
    </filter>

    <filter **>
        @type record_transformer
        @id filter_fluentd_stream_transformer
        <record>
            stream_name ${tag_parts[3]}
        </record>
    </filter>

    <match **>
        @type relabel
        @label @NORMAL
    </match>
</label>
```

To remove Kubernetes metadata from being appended to log events that are sent to CloudWatch, add one line to the `record_transformer` section in the `fluentd.yaml` file. In the log source where you want to remove this metadata, add the following line.

```
remove_keys $.kubernetes.pod_id, $.kubernetes.master_url, $.kubernetes.container_image_id,  
$.kubernetes.namespace_id
```

For example:

```
<filter **>  
  @type record_transformer  
  @id filter_containers_stream_transformer  
  <record>  
    stream_name ${tag_parts[3]}  
  </record>  
  remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,  
  $.kubernetes.container_image_id, $.kubernetes.namespace_id  
</filter>
```

Troubleshooting

If you don't see these log groups and are looking in the correct Region, check the logs for the FluentD DaemonSet pods to look for the error.

Run the following command and make sure that the status is `Running`.

```
kubectl get pods -n amazon-cloudwatch
```

In the results of the previous command, note the pod name that starts with `fluentd-cloudwatch`. Use this pod name in the following command.

```
kubectl logs pod_name -n amazon-cloudwatch
```

If the logs have errors related to IAM permissions, check the IAM role attached to the cluster nodes. For more information about the permissions required to run an Amazon EKS cluster, see [Amazon EKS IAM Policies, Roles, and Permissions](#) in the [Amazon EKS User Guide](#).

If the pod status is `CreateContainerConfigError`, get the exact error by running the following command.

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

If the pod status is `CrashLoopBackOff`, make sure that the architecture of the Fluentd container image is the same as the node when you installed Fluentd. If your cluster has both x86 and ARM64 nodes, you can use a `kubernetes.io/arch` label to place the images on the correct node. For more information, see [kubernetes.io/arch](#).

(Optional) Set up Amazon EKS control plane logging

If you're using Amazon EKS, you can optionally enable Amazon EKS control plane logging, to provide audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs. For more information, see [Amazon EKS Control Plane Logging](#).

(Optional) Enable App Mesh Envoy access logs

You can set up Container Insights FluentD to send App Mesh Envoy access logs to CloudWatch Logs. For more information, see [Access logs](#).

To have Envoy access logs sent to CloudWatch Logs

1. Set up FluentD in the cluster. For more information, see [\(Optional\) Set up FluentD as a DaemonSet to send logs to CloudWatch Logs \(p. 335\)](#).
2. Configure Envoy access logs for your virtual nodes. For instructions, see [Configure Envoy access logs](#). Be sure to configure the log path to be `/dev/stdout` in each virtual node.

When you have finished, the envoy access logs are sent to the `/aws/containerinsights/Cluster_Name/application` log group.

(Optional) Enable the Use_Kubelet feature for large clusters

By default, the Use_Kubelet feature is disabled in the FluentBit Kubernetes plugin. Enabling this feature can reduce traffic to the API server and mitigate the issue of the API Server being a bottleneck. We recommend that you enable this feature for large clusters.

To enable Use_Kubelet, first add the nodes and nodes/proxy permissions to the clusterRole config.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluent-bit-role
rules:
  - nonResourceURLs:
    - /metrics
    verbs:
    - get
  - apiGroups: [""]
    resources:
    - namespaces
    - pods
    - pods/logs
    - nodes
    - nodes/proxy
    verbs: ["get", "list", "watch"]
```

In the DaemonSet configuration, this feature needs host network access. The image version for `amazon/aws-for-fluent-bit` should 2.12.0 or later, or the fluent bit image version should be 1.7.2 or later.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit
  template:
    metadata:
      labels:
        k8s-app: fluent-bit
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
```

```
- name: fluent-bit
  image: amazon/aws-for-fluent-bit:2.19.0
  imagePullPolicy: Always
  env:
    - name: AWS_REGION
      valueFrom:
        configMapKeyRef:
          name: fluent-bit-cluster-info
          key: logs.region
    - name: CLUSTER_NAME
      valueFrom:
        configMapKeyRef:
          name: fluent-bit-cluster-info
          key: cluster.name
    - name: HTTP_SERVER
      valueFrom:
        configMapKeyRef:
          name: fluent-bit-cluster-info
          key: http.server
    - name: HTTP_PORT
      valueFrom:
        configMapKeyRef:
          name: fluent-bit-cluster-info
          key: http.port
    - name: READ_FROM_HEAD
      valueFrom:
        configMapKeyRef:
          name: fluent-bit-cluster-info
          key: read.head
    - name: READ_FROM_TAIL
      valueFrom:
        configMapKeyRef:
          name: fluent-bit-cluster-info
          key: read.tail
    - name: HOST_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: HOSTNAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.name
    - name: CI_VERSION
      value: "k8s/1.3.8"
  resources:
    limits:
      memory: 200Mi
    requests:
      cpu: 500m
      memory: 100Mi
  volumeMounts:
    # Please don't change below read-only permissions
    - name: fluentbitstate
      mountPath: /var/fluent-bit/state
    - name: varlog
      mountPath: /var/log
      readOnly: true
    - name: varlibdockercontainers
      mountPath: /var/lib/docker/containers
      readOnly: true
    - name: fluent-bit-config
      mountPath: /fluent-bit/etc/
    - name: runlogjournal
      mountPath: /run/log/journal
      readOnly: true
```

```
- name: dmesg
  mountPath: /var/log/dmesg
  readOnly: true
terminationGracePeriodSeconds: 10
hostNetwork: true
dnsPolicy: ClusterFirstWithHostNet
volumes:
- name: fluentbitstate
  hostPath:
    path: /var/fluent-bit/state
- name: varlog
  hostPath:
    path: /var/log
- name: varlibdockercontainers
  hostPath:
    path: /var/lib/docker/containers
- name: fluent-bit-config
  configMap:
    name: fluent-bit-config
- name: runlogjournal
  hostPath:
    path: /run/log/journal
- name: dmesg
  hostPath:
    path: /var/log/dmesg
serviceAccountName: fluent-bit
tolerations:
- key: node-role.kubernetes.io/master
  operator: Exists
  effect: NoSchedule
- operator: "Exists"
  effect: "NoExecute"
- operator: "Exists"
  effect: "NoSchedule"
```

The Kubernetes Plugin configuration should be similar to the following:

```
[FILTER]
Name           kubernetes
Match          application./*
Kube_URL       https://kubernetes.default.svc:443
Kube_Tag_Prefix application.var.log.containers.
Merge_Log      On
Merge_Log_Key  log_processed
K8S-Logging.Parser On
K8S-Logging.Exclude Off
Labels         Off
Annotations    Off
Use_Kubelet    On
Kubelet_Port   10250
Buffer_Size    0
```

Updating or deleting Container Insights on Amazon EKS and Kubernetes

Use the steps in these sections to update your CloudWatch agent container image, or to remove Container Insights from an Amazon EKS or Kubernetes cluster.

Topics

- [Updating the CloudWatch agent container image \(p. 343\)](#)
- [Deleting the CloudWatch agent and FluentD for Container Insights \(p. 344\)](#)

Updating the CloudWatch agent container image

If you need to update your container image to the latest version, use the steps in this section.

To update your container image

1. Apply the latest `cwagent-serviceaccount.yaml` file by entering the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-serviceaccount.yaml
```

2. This step is necessary only for customers who upgraded their containerized CloudWatch agent from a version earlier than 1.226589.0, which was released on August 20, 2019.

In the Configmap file `cwagentconfig`, change the keyword `structuredlogs` to `logs`

- a. First, open the existing `cwagentconfig` in edit mode by entering the following command.

```
kubectl edit cm cwagentconfig -n amazon-cloudwatch
```

In the file, if you see the keyword `structuredlogs`, change it to `logs`

- b. Enter `wq` to save the file and exit edit mode.

3. Apply the latest `cwagent-daemonset.yaml` file by entering the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

You can achieve rolling updates of the CloudWatch agent DaemonSet anytime that you change your configuration in `cwagent-configmap.yaml`. To do so, you must make sure the `.spec.template` section in the `cwagent-daemonset.yaml` file has changes. Otherwise, Kubernetes treats the DaemonSet as unchanged. A common practice is to add the hash value of the ConfigMap into `.spec.template.metadata.annotations.configHash`, as in the following example.

```
yq w -i cwagent-daemonset.yaml spec.template.metadata.annotations.configHash $(kubectl get cm/cwagentconfig -n amazon-cloudwatch -o yaml | sha256sum)
```

This adds a hash value into the `cwagent-daemonset.yaml` file, as in the following example.

```
spec:  
  selector:  
    matchLabels:  
      name: cloudwatch-agent  
  template:  
    metadata:  
      labels:  
        name: cloudwatch-agent  
      annotations:  
        configHash: 88915de4cf9c3551a8dc74c0137a3e83569d28c71044b0359c2578d2e0461825
```

Then if you run the following command, the new configuration is picked up.

```
kubectl apply -f cwagent-daemonset.yaml
```

For more information about `yq`, see [yq](#).

Deleting the CloudWatch agent and FluentD for Container Insights

To delete all resources related to the CloudWatch agent and Fluentd, enter the following command. In this command, `Cluster` is the name of your Amazon EKS or Kubernetes cluster, and `Region` is the name of the Region where the logs are published.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluentd-quickstart.yaml | sed "s/{{cluster_name}}/{Cluster_Name}/;s/{{region_name}}/{Region}/" | kubectl delete -f -
```

Viewing Container Insights metrics

After you have Container Insights set up and it is collecting metrics, you can view those metrics in the CloudWatch console.

For Container Insights metrics to appear on your dashboard, you must complete the Container Insights setup. For more information, see [Setting up Container Insights \(p. 307\)](#).

This procedure explains how to view the metrics that Container Insights automatically generates from the collected log data. The rest of this section explains how to further dive into your data and use CloudWatch Logs Insights to see more metrics at more levels of granularity.

To view Container Insights metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Performance Monitoring**.
3. Use the drop-down boxes near the top to select the type of resource to view, as well as the specific resource.

You can set a CloudWatch alarm on any metric that Container Insights collects. For more information, see [Using Amazon CloudWatch alarms \(p. 130\)](#)

Note

If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard. If you have not already enabled Application Insights, you can do so by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.

For more information about Application Insights and containerized applications, see [Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring \(p. 620\)](#).

Viewing the top contributors

For some of the views in Container Insights performance monitoring, you can also see the top contributors by memory or CPU, or the most recently active resources. This is available when you select any of the following dashboards in the drop-down box near the top of the page:

- ECS Services
- ECS Tasks
- EKS Namespaces
- EKS Services
- EKS Pods

When you are viewing one of these types of resources, the bottom of the page displays a table sorted initially by CPU usage. You can change it to sort by memory usage or recent activity. To see more about one of the rows in the table, you can select the checkbox next to that row and then choose **Actions** and choose one of the options in the **Actions** menu.

Using CloudWatch Logs Insights to view Container Insights data

Container Insights collects metrics by using performance log events with using [embedded metric format \(p. 761\)](#). The logs are stored in CloudWatch Logs. CloudWatch generates several metrics automatically from the logs which you can view in the CloudWatch console. You can also do a deeper analysis of the performance data that is collected by using CloudWatch Logs Insights queries.

For more information about CloudWatch Logs Insights, see [Analyze Log Data with CloudWatch Logs Insights](#). For more information about the log fields you can use in queries, see [Container Insights performance log events for Amazon EKS and Kubernetes \(p. 357\)](#).

To use CloudWatch Logs Insights to query your container metric data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Insights**.

Near the top of the screen is the query editor. When you first open CloudWatch Logs Insights, this box contains a default query that returns the 20 most recent log events.

3. In the box above the query editor, select one of the Container Insights log groups to query. For the following example queries to work, the log group name must end with **performance**.

When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in **Discovered fields** in the right pane. It also displays a bar graph of log events in this log group over time. This bar graph shows the distribution of events in the log group that matches your query and time range, not only the events displayed in the table.

4. In the query editor, replace the default query with the following query and choose **Run query**.

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

This query shows a list of nodes, sorted by average node CPU utilization.

5. To try another example, replace that query with another query and choose **Run query**. More sample queries are listed later on this page.

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName
| SORT avg_number_of_container_restarts DESC
```

This query displays a list of your pods, sorted by average number of container restarts.

6. If you want to try another query, you can use include fields in the list at the right of the screen. For more information about query syntax, see [CloudWatch Logs Insights Query Syntax](#).

To see lists of your resources

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Resources**.
3. The default view is a list of your resources being monitored by Container Insights, and alarms that you have set on these resources. To see a visual map of the resources, choose **Map view**.

- From the map view, you can pause your pointer over any resource in the map to see basic metrics about that resource. You can choose any resource to see more detailed graphs about the resource.

Use case: Seeing task-level metrics in Amazon ECS containers

The following example illustrates how to use CloudWatch Logs Insights to dive deeper into your Container Insights logs. For more examples, see the blog [Introducing Amazon CloudWatch Container Insights for Amazon ECS](#).

Container Insights does not automatically generate metrics at the Task level of granularity. The following query displays task-level metrics for CPU and memory usage.

```
stats avg(CpuUtilized) as CPU, avg(MemoryUtilized) as Mem by TaskId, ContainerName  
| sort Mem, CPU desc
```

Other sample queries for Container Insights

List of your pods, sorted by average number of container restarts

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName  
| SORT avg_number_of_container_restarts DESC
```

Pods requested vs. pods running

```
fields @timestamp, @message  
| sort @timestamp desc  
| filter Type="Pod"  
| stats min(pod_number_of_containers) as requested, min(pod_number_of_running_containers)  
as running, ceil(avg(pod_number_of_containers-pod_number_of_running_containers)) as  
pods_missing by kubernetes.pod_name  
| sort pods_missing desc
```

Count of cluster node failures

```
stats avg(cluster_failed_node_count) as CountOfNodeFailures  
| filter Type="Cluster"  
| sort @timestamp desc
```

Application log errors by container name

```
stats count() as countoferrors by kubernetes.container_name  
| filter stream="stderr"  
| sort countoferrors desc
```

Disk usage by container name

```
stats floor(avg(container_filesystem_usage/1024)) as container_filesystem_usage_avg_kb by  
InstanceId, kubernetes.container_name, device  
| filter Type="ContainerFS"  
| sort container_filesystem_usage_avg_kb desc
```

CPU usage by container name

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
```

```
| filter Type="Container"
```

Metrics collected by Container Insights

Container Insights collects one set of metrics for Amazon ECS and AWS Fargate, and a different set for Amazon EKS and Kubernetes.

Topics

- [Amazon ECS Container Insights metrics \(p. 347\)](#)
- [Amazon EKS and Kubernetes Container Insights metrics \(p. 351\)](#)

Amazon ECS Container Insights metrics

The following table lists the metrics and dimensions that Container Insights collects for Amazon ECS. These metrics are in the `ECS/ContainerInsights` namespace. For more information, see [Metrics \(p. 3\)](#).

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see [Setting up Container Insights \(p. 307\)](#).

When you use Container Insights to collect the following metrics, the metrics are charged as custom metrics. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#). Amazon ECS also automatically sends several free metrics to CloudWatch. For more information, see [Available metrics and dimensions](#).

The following metrics are available when you complete the steps in [Setting up Container Insights on Amazon ECS for cluster- and service-level metrics \(p. 307\)](#)

Metric name	Dimensions	Description
ContainerInstanceCount	ClusterName	<p>The number of EC2 instances running the Amazon ECS agent that are registered with a cluster.</p> <p>Unit: Count</p>
CpuUtilized	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	<p>The CPU units used by tasks in the resource that is specified by the dimension set that you're using.</p> <p>This metric is collected only for tasks that have a defined CPU reservation in their task definition.</p>
CpuReserved	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	The CPU units reserved by tasks in the resource that is specified by the

Metric name	Dimensions	Description
		<p>dimension set that you're using.</p> <p>This metric is collected only for tasks that have a defined CPU reservation in their task definition.</p>
DeploymentCount	ServiceName, ClusterName	<p>The number of deployments in an Amazon ECS service.</p>
DesiredTaskCount	ServiceName, ClusterName	<p>The desired number of tasks for an Amazon ECS service.</p> <p>Unit: Count</p>
MemoryUtilized	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	<p>The memory being used by tasks in the resource that is specified by the dimension set that you're using.</p> <p>This metric is collected only for tasks that have a defined memory reservation in their task definition.</p>
MemoryReserved	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	<p>The memory that is reserved by tasks in the resource that is specified by the dimension set that you're using.</p> <p>This metric is collected only for tasks that have a defined memory reservation in their task definition.</p>
NetworkRxBytes	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	<p>The number of bytes received by the resource that is specified by the dimensions that you're using.</p> <p>This metric is available only for containers in tasks using the awsvpc or bridge network modes.</p> <p>Unit: Bytes/Second</p>

Metric name	Dimensions	Description
NetworkTxBytes	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	The number of bytes transmitted by the resource that is specified by the dimensions that you're using. This metric is available only for containers in tasks using the <code>awsvpc</code> or <code>bridge</code> network modes. Unit: Bytes/Second
PendingTaskCount	ServiceName, ClusterName	The number of tasks currently in the <code>PENDING</code> state. Unit: Count
RunningTaskCount	ServiceName, ClusterName	The number of tasks currently in the <code>RUNNING</code> state. Unit: Count
ServiceCount	ClusterName	The number of services in the cluster. Unit: Count
StorageReadBytes	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	The number of bytes read from storage in the resource that is specified by the dimensions that you're using. Unit: Bytes
StorageWriteBytes	TaskDefinitionFamily, ClusterName ServiceName, ClusterName ClusterName	The number of bytes written to storage in the resource that is specified by the dimensions that you're using. Unit: Bytes
TaskCount	ClusterName	The number of tasks running in the cluster. Unit: Count

Metric name	Dimensions	Description
TaskSetCount	ServiceName, ClusterName	The number of task sets in the service. Unit: Count

The following metrics are available when you complete the steps in [Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS \(p. 311\)](#)

Metric name	Dimensions	Description
instance_cpu_limit	ClusterName	The maximum number of CPU units that can be assigned to a single EC2 Instance in the cluster.
instance_cpu_reserve	InstanceId, ContainerInstanceId, ClusterName	The percentage of CPU currently being reserved on a single EC2 instance in the cluster.
instance_cpu_usage	ClusterName	The number of CPU units being used on a Single EC2 instance in the cluster.
instance_cpu_utilization	InstanceId, ContainerInstanceId, ClusterName	The total percentage of CPU units being used on a single EC2 instance in the cluster.
instance_filesystem	ClusterName	The total percentage of file system capacity being used on a single EC2 instance in the cluster.
instance_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single EC2 Instance in this cluster.
instance_memory_reserve	InstanceId, ContainerInstanceId, ClusterName	The percentage of Memory currently being reserved on a single EC2 Instance in the cluster.
instance_memory_utilization	ClusterName	The total percentage of memory being used on a single EC2 Instance in the cluster.

Metric name	Dimensions	Description
instance_memory_working	ClusterName	The amount of memory, in bytes, being used on a single EC2 Instance in the cluster.
instance_network_total	ClusterName	The total number of bytes per second transmitted and received over the network on a single EC2 Instance in the cluster.
instance_number_of_running_tasks	ClusterName	The number of running tasks on a single EC2 Instance in the cluster.

Amazon EKS and Kubernetes Container Insights metrics

The following table lists the metrics and dimensions that Container Insights collects for Amazon EKS and Kubernetes. These metrics are in the `ContainerInsights` namespace. For more information, see [Metrics \(p. 3\)](#).

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see [Setting up Container Insights \(p. 307\)](#).

When you use Container Insights to collect the following metrics, the metrics are charged as custom metrics. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

Metric name	Dimensions	Description
cluster_failed_node	ClusterName	The number of failed worker nodes in the cluster. A node is considered failed if it is suffering from any <i>node conditions</i> . For more information, see Conditions in the Kubernetes documentation.
cluster_node_count	ClusterName	The total number of worker nodes in the cluster.
namespace_number_of_pods	NamespaceName, ClusterName	The number of pods running per namespace in the resource that is specified by the dimensions that you're using.

Metric name	Dimensions	Description
node_cpu_limit	ClusterName	The maximum number of CPU units that can be assigned to a single node in this cluster.
node_cpu_reserved_capacity	NodeName, ClusterName, Instanceld ClusterName	The percentage of CPU units that are reserved for node components, such as kubelet, kube-proxy, and Docker.
node_cpu_usage_total	ClusterName	The number of CPU units being used on the nodes in the cluster.
node_cpu_utilization	NodeName, ClusterName, Instanceld ClusterName	The total percentage of CPU units being used on the nodes in the cluster.
node_filesystem_utilization	NodeName, ClusterName, Instanceld ClusterName	The total percentage of file system capacity being used on nodes in the cluster.
node_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single node in this cluster.
node_memory_reserved	NodeName, ClusterName, Instanceld ClusterName	The percentage of memory currently being used on the nodes in the cluster.
node_memory_utilization	NodeName, ClusterName, Instanceld ClusterName	The percentage of memory currently being used by the node or nodes. node_memory_utilization is calculated by node_memory_working_set /node_memory_limit. It is the percentage of node memory usage over the node memory limitation.
node_memory_working_set	ClusterName	The amount of memory, in bytes, being used in the working set of the nodes in the cluster.

Metric name	Dimensions	Description
node_network_total_bytes	NodeName, ClusterName, InstanceId ClusterName	The total number of bytes per second transmitted and received over the network per node in a cluster.
node_number_of_running_containers	NodeName, ClusterName, InstanceId ClusterName	The number of running containers per node in a cluster.
node_number_of_running_pods	NodeName, ClusterName, InstanceId ClusterName	The number of running pods per node in a cluster.
pod_cpu_reserved_capacity	PodName, Namespace, ClusterName ClusterName	The CPU capacity that is reserved per pod in a cluster.
pod_cpu_utilization	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	The percentage of CPU units being used by pods.
pod_cpu_utilization_over_limit	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	The percentage of CPU units being used by pods that is over the pod limit.
pod_memory_reserved_percent	PodName, Namespace, ClusterName ClusterName	The percentage of memory that is reserved for pods.
pod_memory_utilization_percent	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	The percentage of memory currently being used by the pod or pods.
pod_memory_utilization_over_limit_percent	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	The percentage of memory that is being used by pods that is over the pod limit.
pod_number_of_container_restarts	PodName, Namespace, ClusterName	The total number of container restarts in a pod.

Metric name	Dimensions	Description
pod_network_rx_bytes	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	The number of bytes per second being received over the network by the pod.
pod_network_tx_bytes	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	The number of bytes per second being transmitted over the network by the pod.
service_number_of_running_pods	Service, Namespace, ClusterName ClusterName	The number of pods running the service or services in the cluster.

Container Insights performance log reference

This section includes reference information about how Container Insights uses performance log events to collect metrics.

Topics

- [Container Insights performance log events for Amazon ECS \(p. 354\)](#)
- [Container Insights performance log events for Amazon EKS and Kubernetes \(p. 357\)](#)
- [Relevant fields in performance log events for Amazon EKS and Kubernetes \(p. 368\)](#)

Container Insights performance log events for Amazon ECS

The following are examples of the performance log events that Container Insights collects from Amazon ECS.

Type: Container

```
{
  "Version": "0",
  "Type": "Container",
  "ContainerName": "sleep",
  "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
  "TaskDefinitionFamily": "sleep360",
  "TaskDefinitionRevision": "1",
  "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
  "EC2InstanceId": "i-0c470579dbcdbd2f3",
  "ClusterName": "MyCluster",
  "Image": "busybox",
  "ContainerKnownStatus": "RUNNING",
  "Timestamp": 1623963900000,
  "CpuUtilized": 0.0,
```

```
    "CpuReserved":10.0,
    "MemoryUtilized":0,
    "MemoryReserved":10,
    "StorageReadBytes":0,
    "StorageWriteBytes":0,
    "NetworkRxBytes":0,
    "NetworkRxDropped":0,
    "NetworkRxErrors":0,
    "NetworkRxPackets":14,
    "NetworkTxBytes":0,
    "NetworkTxDropped":0,
    "NetworkTxErrors":0,
    "NetworkTxPackets":0
}
```

Type: Task

```
{
  "Version": "0",
  "Type": "Task",
  "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
  "TaskDefinitionFamily": "sleep360",
  "TaskDefinitionRevision": "1",
  "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
  "EC2InstanceId": "i-0c470579dbcdbd2f3",
  "ClusterName": "MyCluster",
  "AccountID": "637146863587",
  "Region": "us-west-2",
  "AvailabilityZone": "us-west-2b",
  "KnownStatus": "RUNNING",
  "LaunchType": "EC2",
  "PullStartedAt": 1623963608201,
  "PullStoppedAt": 1623963610065,
  "CreatedAt": 1623963607094,
  "StartedAt": 1623963610382,
  "Timestamp": 1623963900000,
  "CpuUtilized": 0.0,
  "CpuReserved": 10.0,
  "MemoryUtilized": 0,
  "MemoryReserved": 10,
  "StorageReadBytes": 0,
  "StorageWriteBytes": 0,
  "NetworkRxBytes": 0,
  "NetworkRxDropped": 0,
  "NetworkRxErrors": 0,
  "NetworkRxPackets": 14,
  "NetworkTxBytes": 0,
  "NetworkTxDropped": 0,
  "NetworkTxErrors": 0,
  "NetworkTxPackets": 0,
  "CloudWatchMetrics": [
    {
      "Namespace": "ECS/ContainerInsights",
      "Metrics": [
        {
          "Name": "CpuUtilized",
          "Unit": "None"
        },
        {
          "Name": "CpuReserved",
          "Unit": "None"
        },
        {
          "Name": "MemoryUtilized",
          "Unit": "Megabytes"
        }
      ]
    }
  ]
}
```

```

        "Name": "MemoryReserved",
        "Unit": "Megabytes"
    },
    {
        "Name": "StorageReadBytes",
        "Unit": "Bytes/Second"
    },
    {
        "Name": "StorageWriteBytes",
        "Unit": "Bytes/Second"
    },
    {
        "Name": "NetworkRxBytes",
        "Unit": "Bytes/Second"
    },
    {
        "Name": "NetworkTxBytes",
        "Unit": "Bytes/Second"
    }
],
"Dimensions": [
    [
        "ClusterName"
    ],
    [
        "ClusterName",
        "TaskDefinitionFamily"
    ]
]
}
}

```

Type: Service

```

{
    "Version": "0",
    "Type": "Service",
    "ServiceName": "myCIService",
    "ClusterName": "myCICluster",
    "Timestamp": 1561586460000,
    "DesiredTaskCount": 2,
    "RunningTaskCount": 2,
    "PendingTaskCount": 0,
    "DeploymentCount": 1,
    "TaskSetCount": 0,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "DesiredTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "RunningTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "PendingTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "DeploymentCount",
                    "Unit": "Count"
                }
            ]
        }
    ]
}

```

```
        {
            "Name": "TaskSetCount",
            "Unit": "Count"
        }
    ],
    "Dimensions": [
        [
            "ServiceName",
            "ClusterName"
        ]
    ]
}
]
```

Type: Cluster

```
{
    "Version": "0",
    "Type": "Cluster",
    "ClusterName": "myCICluster",
    "Timestamp": 1561587300000,
    "TaskCount": 5,
    "ContainerInstanceCount": 5,
    "ServiceCount": 2,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "TaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ContainerInstanceCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ServiceCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Container Insights performance log events for Amazon EKS and Kubernetes

The following are examples of the performance log events that Container Insights collects from Amazon EKS and Kubernetes clusters.

Type: Node

```
{
```

```
"AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-  
NodeGroup-1174PV2WHZAYU",  
"CloudWatchMetrics": [  
  {  
    "Metrics": [  
      {  
        "Unit": "Percent",  
        "Name": "node_cpu_utilization"  
      },  
      {  
        "Unit": "Percent",  
        "Name": "node_memory_utilization"  
      },  
      {  
        "Unit": "Bytes/Second",  
        "Name": "node_network_total_bytes"  
      },  
      {  
        "Unit": "Percent",  
        "Name": "node_cpu_reserved_capacity"  
      },  
      {  
        "Unit": "Percent",  
        "Name": "node_memory_reserved_capacity"  
      },  
      {  
        "Unit": "Count",  
        "Name": "node_number_of_running_pods"  
      },  
      {  
        "Unit": "Count",  
        "Name": "node_number_of_running_containers"  
      }  
    ],  
    "Dimensions": [  
      [  
        "NodeName",  
        "InstanceId",  
        "ClusterName"  
      ]  
    ],  
    "Namespace": "ContainerInsights"  
  },  
  {  
    "Metrics": [  
      {  
        "Unit": "Percent",  
        "Name": "node_cpu_utilization"  
      },  
      {  
        "Unit": "Percent",  
        "Name": "node_memory_utilization"  
      },  
      {  
        "Unit": "Bytes/Second",  
        "Name": "node_network_total_bytes"  
      },  
      {  
        "Unit": "Percent",  
        "Name": "node_cpu_reserved_capacity"  
      },  
      {  
        "Unit": "Percent",  
        "Name": "node_memory_reserved_capacity"  
      },  
      {  
        "Unit": "Count",  
        "Name": "node_number_of_running_pods"  
      },  
      {  
        "Unit": "Count",  
        "Name": "node_number_of_running_containers"  
      }  
    ]  
  }  
]
```

```
        "Unit": "Count",
        "Name": "node_number_of_running_pods"
    },
    {
        "Unit": "Count",
        "Name": "node_number_of_running_containers"
    },
    {
        "Name": "node_cpu_usage_total"
    },
    {
        "Name": "node_cpu_limit"
    },
    {
        "Unit": "Bytes",
        "Name": "node_memory_working_set"
    },
    {
        "Unit": "Bytes",
        "Name": "node_memory_limit"
    }
],
"Dimensions": [
    [
        "ClusterName"
    ]
],
"Namespace": "ContainerInsights"
},
],
"ClusterName": "myCICluster",
"InstanceId": "i-1234567890123456",
"InstanceType": "t3.xlarge",
"NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
"Sources": [
    "cadvisor",
    "/proc",
    "pod",
    "calculated"
],
"Timestamp": "1567096682364",
>Type": "Node",
"Version": "0",
"kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
},
"node_cpu_limit": 4000,
"node_cpu_request": 1130,
"node_cpu_reserved_capacity": 28.249999999999996,
"node_cpu_usage_system": 33.794636630852764,
"node_cpu_usage_total": 136.47852169244098,
"node_cpu_usage_user": 71.67075111567326,
"node_cpu_utilization": 3.4119630423110245,
"node_memory_cache": 3103297536,
"node_memory_failcnt": 0,
"node_memory_hierarchical_pgfault": 0,
"node_memory_hierarchical_pgmajfault": 0,
"node_memory_limit": 16624865280,
"node_memory_mapped_file": 406646784,
"node_memory_max_usage": 4230746112,
"node_memory_pgfault": 0,
"node_memory_pgmajfault": 0,
"node_memory_request": 1115684864,
"node_memory_reserved_capacity": 6.7109407818311055,
"node_memory_rss": 798146560,
"node_memory_swap": 0,
```

```

    "node_memory_usage": 3901444096,
    "node_memory_utilization": 6.601302600149552,
    "node_memory_working_set": 1097457664,
    "node_network_rx_bytes": 35918.392817386324,
    "node_network_rx_dropped": 0,
    "node_network_rx_errors": 0,
    "node_network_rx_packets": 157.67565245448117,
    "node_network_total_bytes": 68264.20276554905,
    "node_network_tx_bytes": 32345.80994816272,
    "node_network_tx_dropped": 0,
    "node_network_tx_errors": 0,
    "node_network_tx_packets": 154.21455923431654,
    "node_number_of_running_containers": 16,
    "node_number_of_running_pods": 13
}

```

Type: NodeFS

```

{
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
    "CloudWatchMetrics": [
        {
            "Metrics": [
                {
                    "Unit": "Percent",
                    "Name": "node_filesystem_utilization"
                }
            ],
            "Dimensions": [
                [
                    "NodeName",
                    "InstanceId",
                    "ClusterName"
                ],
                [
                    "ClusterName"
                ]
            ],
            "Namespace": "ContainerInsights"
        }
    ],
    "ClusterName": "myCICluster",
    "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
    "InstanceId": "i-1234567890123456",
    "InstanceType": "t3.xlarge",
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
    "Sources": [
        "cadvisor",
        "calculated"
    ],
    "Timestamp": "1567097939726",
    "Type": "NodeFS",
    "Version": "0",
    "device": "/dev/nvme0n1p1",
    "fstype": "vfat",
    "kubernetes": {
        "host": "ip-192-168-75-26.us-west-2.compute.internal"
    },
    "node_filesystem_available": 17298395136,
    "node_filesystem_capacity": 21462233088,
    "node_filesystem_inodes": 10484720,
    "node_filesystem_inodes_free": 10367158,
    "node_filesystem_usage": 4163837952,
    "node_filesystem_utilization": 19.400767547940255
}

```

```
}
```

Type: NodeDiskIO

```
{  
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-  
NodeGroup-1174PV2WHZAYU",  
    "ClusterName": "myCICluster",  
    "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",  
    "InstanceId": "i-1234567890123456",  
    "InstanceType": "t3.xlarge",  
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",  
    "Sources": [  
        "cadvisor"  
    ],  
    "Timestamp": "1567096928131",  
    "Type": "NodeDiskIO",  
    "Version": "0",  
    "device": "/dev/nvme0n1",  
    "kubernetes": {  
        "host": "ip-192-168-75-26.us-west-2.compute.internal"  
    },  
    "node_diskio_io_service_bytes_async": 9750.505814277016,  
    "node_diskio_io_service_bytes_read": 0,  
    "node_diskio_io_service_bytes_sync": 230.6174506688036,  
    "node_diskio_io_service_bytes_total": 9981.123264945818,  
    "node_diskio_io_service_bytes_write": 9981.123264945818,  
    "node_diskio_io_serviced_async": 1.153087253344018,  
    "node_diskio_io_serviced_read": 0,  
    "node_diskio_io_serviced_sync": 0.03603397666700056,  
    "node_diskio_io_serviced_total": 1.1891212300110185,  
    "node_diskio_io_serviced_write": 1.1891212300110185  
}
```

Type: NodeNet

```
{  
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-  
NodeGroup-1174PV2WHZAYU",  
    "ClusterName": "myCICluster",  
    "InstanceId": "i-1234567890123456",  
    "InstanceType": "t3.xlarge",  
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",  
    "Sources": [  
        "cadvisor",  
        "calculated"  
    ],  
    "Timestamp": "1567096928131",  
    "Type": "NodeNet",  
    "Version": "0",  
    "interface": "eni972f6bfa9a0",  
    "kubernetes": {  
        "host": "ip-192-168-75-26.us-west-2.compute.internal"  
    },  
    "node_interface_network_rx_bytes": 3163.008420864309,  
    "node_interface_network_rx_dropped": 0,  
    "node_interface_network_rx_errors": 0,  
    "node_interface_network_rx_packets": 16.575629266820258,  
    "node_interface_network_total_bytes": 3518.3935157426017,  
    "node_interface_network_tx_bytes": 355.385094878293,  
    "node_interface_network_tx_dropped": 0,  
    "node_interface_network_tx_errors": 0,  
    "node_interface_network_tx_packets": 3.9997714100370625  
}
```

```
| }
```

Type: Pod

```
{  
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-  
NodeGroup-1174PV2WHZAYU",  
    "CloudWatchMetrics": [  
        {  
            "Metrics": [  
                {  
                    "Unit": "Percent",  
                    "Name": "pod_cpu_utilization"  
                },  
                {  
                    "Unit": "Percent",  
                    "Name": "pod_memory_utilization"  
                },  
                {  
                    "Unit": "Bytes/Second",  
                    "Name": "pod_network_rx_bytes"  
                },  
                {  
                    "Unit": "Bytes/Second",  
                    "Name": "pod_network_tx_bytes"  
                },  
                {  
                    "Unit": "Percent",  
                    "Name": "pod_cpu_utilization_over_pod_limit"  
                },  
                {  
                    "Unit": "Percent",  
                    "Name": "pod_memory_utilization_over_pod_limit"  
                }  
            ],  
            "Dimensions": [  
                [  
                    "PodName",  
                    "Namespace",  
                    "ClusterName"  
                ],  
                [  
                    "Service",  
                    "Namespace",  
                    "ClusterName"  
                ],  
                [  
                    "Namespace",  
                    "ClusterName"  
                ],  
                [  
                    "ClusterName"  
                ]  
            ],  
            "Namespace": "ContainerInsights"  
        },  
        {  
            "Metrics": [  
                {  
                    "Unit": "Percent",  
                    "Name": "pod_cpu_reserved_capacity"  
                },  
                {  
                    "Unit": "Percent",  
                    "Name": "pod_memory_reserved_capacity"  
                }  
            ]  
        }  
    ]  
}
```

```

        },
    ],
    "Dimensions": [
        [
            "PodName",
            "Namespace",
            "ClusterName"
        ],
        [
            "ClusterName"
        ]
    ],
    "Namespace": "ContainerInsights"
},
{
    "Metrics": [
        {
            "Unit": "Count",
            "Name": "pod_number_of_container_restarts"
        }
    ],
    "Dimensions": [
        [
            "PodName",
            "Namespace",
            "ClusterName"
        ]
    ],
    "Namespace": "ContainerInsights"
}
],
"ClusterName": "myCICluster",
"InstanceId": "i-1234567890123456",
"InstanceType": "t3.xlarge",
"Namespace": "amazon-cloudwatch",
"NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
"PodName": "cloudwatch-agent-statsd",
"Service": "cloudwatch-agent-statsd",
"Sources": [
    "cadvisor",
    "pod",
    "calculated"
],
"Timestamp": "1567097351092",
>Type": "Pod",
"Version": "0",
"kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
        "app": "cloudwatch-agent-statsd",
        "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
        {
            "owner_kind": "Deployment",
            "owner_name": "cloudwatch-agent-statsd"
        }
    ],
    "service_name": "cloudwatch-agent-statsd"
},
"pod_cpu_limit": 200,
"pod_cpu_request": 200,
"pod_cpu_reserved_capacity": 5,

```

```

    "pod_cpu_usage_system": 1.4504841104992765,
    "pod_cpu_usage_total": 5.817016867430125,
    "pod_cpu_usage_user": 1.1281543081661038,
    "pod_cpu_utilization": 0.14542542168575312,
    "pod_cpu_utilization_over_pod_limit": 2.9085084337150624,
    "pod_memory_cache": 8192,
    "pod_memory_failcnt": 0,
    "pod_memory_hierarchical_pgfault": 0,
    "pod_memory_hierarchical_pgmajfault": 0,
    "pod_memory_limit": 104857600,
    "pod_memory_mapped_file": 0,
    "pod_memory_max_usage": 25268224,
    "pod_memory_pgfault": 0,
    "pod_memory_pgmajfault": 0,
    "pod_memory_request": 104857600,
    "pod_memory_reserved_capacity": 0.6307275170893897,
    "pod_memory_rss": 22777856,
    "pod_memory_swap": 0,
    "pod_memory_usage": 25141248,
    "pod_memory_utilization": 0.10988455961791709,
    "pod_memory_utilization_over_pod_limit": 17.421875,
    "pod_memory_working_set": 18268160,
    "pod_network_rx_bytes": 9880.697124714186,
    "pod_network_rx_dropped": 0,
    "pod_network_rx_errors": 0,
    "pod_network_rx_packets": 107.80005532263283,
    "pod_network_total_bytes": 10158.829201483635,
    "pod_network_tx_bytes": 278.13207676944796,
    "pod_network_tx_dropped": 0,
    "pod_network_tx_errors": 0,
    "pod_network_tx_packets": 1.146027574644318,
    "pod_number_of_container_restarts": 0,
    "pod_number_of_containers": 1,
    "pod_number_of_running_containers": 1,
    "pod_status": "Running"
}

```

Type: PodNet

```
{
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-1174PV2WHZAYU",
    "ClusterName": "myCICluster",
    "InstanceId": "i-1234567890123456",
    "InstanceType": "t3.xlarge",
    "Namespace": "amazon-cloudwatch",
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
    "PodName": "cloudwatch-agent-statsd",
    "Service": "cloudwatch-agent-statsd",
    "Sources": [
        "cadvisor",
        "calculated"
    ],
    "Timestamp": "1567097351092",
    "Type": "PodNet",
    "Version": "0",
    "interface": "eth0",
    "kubernetes": {
        "host": "ip-192-168-75-26.us-west-2.compute.internal",
        "labels": {
            "app": "cloudwatch-agent-statsd",
            "pod-template-hash": "df44f855f"
        }
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
}
```

```

"pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
"pod_owners": [
    {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
    }
],
"service_name": "cloudwatch-agent-statsd"
},
"pod_interface_network_rx_bytes": 9880.697124714186,
"pod_interface_network_rx_dropped": 0,
"pod_interface_network_rx_errors": 0,
"pod_interface_network_rx_packets": 107.80005532263283,
"pod_interface_network_total_bytes": 10158.829201483635,
"pod_interface_network_tx_bytes": 278.13207676944796,
"pod_interface_network_tx_dropped": 0,
"pod_interface_network_tx_errors": 0,
"pod_interface_network_tx_packets": 1.146027574644318
}

```

Type: Container

```

{
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-sample",
    "ClusterName": "myCICluster",
    "InstanceId": "i-1234567890123456",
    "InstanceType": "t3.xlarge",
    "Namespace": "amazon-cloudwatch",
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
    "PodName": "cloudwatch-agent-statsd",
    "Service": "cloudwatch-agent-statsd",
    "Sources": [
        "cadvisor",
        "pod",
        "calculated"
    ],
    "Timestamp": "1567097399912",
    "Type": "Container",
    "Version": "0",
    "container_cpu_limit": 200,
    "container_cpu_request": 200,
    "container_cpu_usage_system": 1.87958283771964,
    "container_cpu_usage_total": 6.159993652997942,
    "container_cpu_usage_user": 1.6707403001952357,
    "container_cpu_utilization": 0.15399984132494854,
    "container_memory_cache": 8192,
    "container_memory_failcnt": 0,
    "container_memory_hierarchical_pgfault": 0,
    "container_memory_hierarchical_pgmajfault": 0,
    "container_memory_limit": 104857600,
    "container_memory_mapped_file": 0,
    "container_memory_max_usage": 24580096,
    "container_memory_pgfault": 0,
    "container_memory_pgmajfault": 0,
    "container_memory_request": 104857600,
    "container_memory_rss": 22736896,
    "container_memory_swap": 0,
    "container_memory_usage": 24453120,
    "container_memory_utilization": 0.10574541028701798,
    "container_memory_working_set": 17580032,
    "container_status": "Running",
    "kubernetes": {
        "container_name": "cloudwatch-agent",
        "docker": {
            "container_id": "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
        }
    }
}

```

```

},
"host": "ip-192-168-75-26.us-west-2.compute.internal",
"labels": {
    "app": "cloudwatch-agent-statsd",
    "pod-template-hash": "df44f855f"
},
"namespace_name": "amazon-cloudwatch",
"pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
"pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
"pod_owners": [
    {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
    }
],
"service_name": "cloudwatch-agent-statsd"
},
"number_of_container_restarts": 0
}

```

Type: ContainerFS

```

{
    "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-1174PV2WHZAYU",
    "ClusterName": "myCICluster",
    "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
    "InstanceId": "i-1234567890123456",
    "InstanceType": "t3.xlarge",
    "Namespace": "amazon-cloudwatch",
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
    "PodName": "cloudwatch-agent-statsd",
    "Service": "cloudwatch-agent-statsd",
    "Sources": [
        "cadvisor",
        "calculated"
    ],
    "Timestamp": "1567097399912",
    "Type": "ContainerFS",
    "Version": "0",
    "container_filesystem_available": 0,
    "container_filesystem_capacity": 21462233088,
    "container_filesystem_usage": 24576,
    "container_filesystem_utilization": 0.0001145081217748071,
    "device": "/dev/nvmeOnp1",
    "fstype": "vfs",
    "kubernetes": {
        "container_name": "cloudwatch-agent",
        "docker": {
            "container_id": "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
        },
        "host": "ip-192-168-75-26.us-west-2.compute.internal",
        "labels": {
            "app": "cloudwatch-agent-statsd",
            "pod-template-hash": "df44f855f"
        },
        "namespace_name": "amazon-cloudwatch",
        "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
        "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
        "pod_owners": [
            {
                "owner_kind": "Deployment",
                "owner_name": "cloudwatch-agent-statsd"
            }
        ],
    }
}

```

```
        "service_name": "cloudwatch-agent-statsd"
    }
}
```

Type: Cluster

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "cluster_node_count"
        },
        {
          "Unit": "Count",
          "Name": "cluster_failed_node_count"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "Cluster",
  "Version": "0",
  "cluster_failed_node_count": 0,
  "cluster_node_count": 3
}
```

Type: ClusterService

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "service_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Timestamp": "1567097534160",
  "Type": "ClusterService",
  "Version": "0"
}
```

```
"Service": "cloudwatch-agent-statsd",
"Sources": [
    "apiserver"
],
"Timestamp": "1567097534160",
>Type": "ClusterService",
"Version": "0",
"kubernetes": {
    "namespace_name": "amazon-cloudwatch",
    "service_name": "cloudwatch-agent-statsd"
},
"service_number_of_running_pods": 1
}
```

Type: ClusterNamespace

```
{
    "CloudWatchMetrics": [
        {
            "Metrics": [
                {
                    "Unit": "Count",
                    "Name": "namespace_number_of_running_pods"
                }
            ],
            "Dimensions": [
                [
                    "Namespace",
                    "ClusterName"
                ],
                [
                    "ClusterName"
                ]
            ],
            "Namespace": "ContainerInsights"
        }
    ],
    "ClusterName": "myCICluster",
    "Namespace": "amazon-cloudwatch",
    "Sources": [
        "apiserver"
    ],
    "Timestamp": "1567097594160",
    "Type": "ClusterNamespace",
    "Version": "0",
    "kubernetes": {
        "namespace_name": "amazon-cloudwatch"
    },
    "namespace_number_of_running_pods": 7
}
```

Relevant fields in performance log events for Amazon EKS and Kubernetes

For Amazon EKS and Kubernetes, the containerized CloudWatch agent emits data as performance log events. This enables CloudWatch to ingest and store high-cardinality data. CloudWatch uses the data in the performance log events to create aggregated CloudWatch metrics at the cluster, node, and pod levels without the need to lose granular details.

The following table lists the fields in these performance log events that are relevant to the collection of Container Insights metric data. You can use CloudWatch Logs Insights to query for any of these fields

to collect data or investigate issues. For more information, see [Analyze Log Data With CloudWatch Logs Insights](#).

Type	Log field	Source	Formula or notes
Pod	pod_cpu_utilization	Calculated	Formula: <code>pod_cpu_usage_total / node_cpu_limit</code>
Pod	pod_cpu_usage_total pod_cpu_usage_total is reported in millicores.	cadvisor	
Pod	pod_cpu_limit	Calculated	Formula: <code>sum(container_cpu_limit)</code> If any containers in the pod don't have a CPU limit defined, this field doesn't appear in the log event.
Pod	pod_cpu_request	Calculated	Formula: <code>sum(container_cpu_request)</code> container_cpu_request isn't guaranteed to be set. Only the ones that are set are included in the sum.
Pod	pod_cpu_utilization_over_pod_limit	Calculated	Formula: <code>pod_cpu_usage_total / pod_cpu_limit</code>
Pod	pod_cpu_reserved_capacity	Calculated	Formula: <code>pod_cpu_request / node_cpu_limit</code>
Pod	pod_memory_utilization	Calculated	Formula: <code>pod_memory_working_set / node_memory_limit</code> It is the percentage of pod memory usage over the node memory limitation.
Pod	pod_memory_working_set	cadvisor	
Pod	pod_memory_limit	Calculated	Formula: <code>sum(container_memory_limit)</code>

Type	Log field	Source	Formula or notes
			If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event.
Pod	pod_memory_request	Calculated	Formula: <code>sum(container_memory_request)</code> container_memory_request isn't guaranteed to be set. Only the ones that are set are included in the sum.
Pod	pod_memory_utilization_over_pod_limit	Calculated	Formula: <code>pod_memory_working_set / pod_memory_limit</code> If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event.
Pod	pod_memory_reserved_capacity	Calculated	Formula: <code>pod_memory_request / node_memory_limit</code>
Pod	pod_network_tx_bytes	Calculated	Formula: <code>sum(pod_interface_network_tx_bytes)</code> This data is available for all the network interfaces per pod. The CloudWatch agent calculates the total and adds metric extraction rules.
Pod	pod_network_rx_bytes	Calculated	Formula: <code>sum(pod_interface_network_rx_bytes)</code>
Pod	pod_network_total_bytes	Calculated	Formula: <code>pod_network_rx_bytes + pod_network_tx_bytes</code>

Type	Log field	Source	Formula or notes
PodNet	pod_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a pod network interface.
PodNet	pod_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a pod network interface.
Container	container_cpu_usage_total	cadvisor	
Container	container_cpu_limit	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_cpu_request	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_working_set	cadvisor	
Container	container_memory_limit	pod	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_request	pod	Not guaranteed to be set. It's not emitted if it's not set.
ContainerFS	container_filesystem_capacity	pod	This data is available per disk device.
ContainerFS	container_filesystem_usage	pod	This data is available per disk device.
ContainerFS	container_filesystem_utilization	Calculated	Formula: <code>container_filesystem_usage / container_filesystem_capacity</code> This data is available per device name.
Node	node_cpu_utilization	Calculated	Formula: <code>node_cpu_usage_total / node_cpu_limit</code>

Amazon CloudWatch User Guide
 Relevant fields in performance log
 events for Amazon EKS and Kubernetes

Type	Log field	Source	Formula or notes
Node	node_cpu_usage_total	cadvisor	
Node	node_cpu_limit	/proc	
Node	node_cpu_request	Calculated	Formula: sum(pod_cpu_request)
Node	node_cpu_reserved_capacity	Calculated	Formula: node_cpu_request / node_cpu_limit
Node	node_memory_utilization	Calculated	Formula: node_memory_working_set / node_memory_limit
Node	node_memory_working_set	cadvisor	
Node	node_memory_limit	/proc	
Node	node_memory_request	Calculated	Formula: sum(pod_memory_request)
Node	node_memory_reserved_capacity	Calculated	Formula: node_memory_request / node_memory_limit
Node	node_network_rx_bytes	Calculated	Formula: sum(node_interface_network_rx_bytes)
Node	node_network_tx_bytes	Calculated	Formula: sum(node_interface_network_tx_bytes)
Node	node_network_total_bytes	Calculated	Formula: node_network_rx_bytes + node_network_tx_bytes
Node	node_number_of_running_pods	Pod List	
Node	node_number_of_running_containers	Pod List	
NodeNet	node_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a worker node network interface.
NodeNet	node_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a worker node network interface.
NodeFS	node_filesystem_capacity	cadvisor	

Type	Log field	Source	Formula or notes
NodeFS	node_filesystem_usage	cadvisor	
NodeFS	node_filesystem_utilization	Calculated	Formula: $\text{node_filesystem_usage} / \text{node_filesystem_capacity}$ This data is available per device name.
Cluster	cluster_failed_node_count	API Server	
Cluster	cluster_node_count	API Server	
Service	service_number_of_running_pods	API Server	
Namespace	namespace_number_of_running_pods	API Server	

Metrics calculation examples

This section includes examples that show how some of the values in the preceding table are calculated.

Suppose that you have a cluster in the following state.

```

Node1
  node_cpu_limit = 4
  node_cpu_usage_total = 3

Pod1
  pod_cpu_usage_total = 2

  Container1
    container_cpu_limit = 1
    container_cpu_request = 1
    container_cpu_usage_total = 0.8

  Container2
    container_cpu_limit = null
    container_cpu_request = null
    container_cpu_usage_total = 1.2

Pod2
  pod_cpu_usage_total = 0.4

  Container3
    container_cpu_limit = 1
    container_cpu_request = 0.5
    container_cpu_usage_total = 0.4

Node2
  node_cpu_limit = 8
  node_cpu_usage_total = 1.5

Pod3
  pod_cpu_usage_total = 1

  Container4
    container_cpu_limit = 2

```

```
container_cpu_request = 2
container_cpu_usage_total = 1
```

The following table shows how pod CPU metrics are calculated using this data.

Metric	Formula	Pod1	Pod2	Pod3
pod_cpu_utilization	$\text{pod_cpu_usage_total} / \text{node_cpu_limit}$	$2 / 4 = 50\%$	$0.4 / 4 = 10\%$	$1 / 8 = 12.5\%$
pod_cpu_utilization_over_pod_limit	$\text{pod_cpu_usage_total} / \text{sum(container_cpu_limit)}$	N/A because CPU limit for Container2 isn't defined	$0.4 / 1 = 40\%$	$1 / 2 = 50\%$
pod_cpu_reserved_capacity	$\text{sum(container_cpu_request)} / \text{node_cpu_limit}$	$(1 + 0) / 4 = 25\%$	$0.5 / 4 = 12.5\%$	$2 / 8 = 25\%$

The following table shows how node CPU metrics are calculated using this data.

Metric	Formula	Node1	Node2
node_cpu_utilization	$\text{node_cpu_usage_total} / \text{node_cpu_limit}$	$3 / 4 = 75\%$	$1.5 / 8 = 18.75\%$
node_cpu_reserved_capacity	$\text{sum(pod_cpu_request)} / \text{node_cpu_limit}$	$1.5 / 4 = 37.5\%$	$2 / 8 = 25\%$

Container Insights Prometheus metrics monitoring

CloudWatch Container Insights monitoring for Prometheus automates the discovery of Prometheus metrics from containerized systems and workloads. Prometheus is an open-source systems monitoring and alerting toolkit. For more information, see [What is Prometheus?](#) in the Prometheus documentation.

Discovering Prometheus metrics is supported for [Amazon Elastic Container Service](#), [Amazon Elastic Kubernetes Service](#) and [Kubernetes](#) clusters running on Amazon EC2 instances. The Prometheus counter, gauge, and summary metric types are collected. Support for histogram metrics is planned for an upcoming release.

For Amazon ECS and Amazon EKS clusters, both the EC2 and Fargate launch types are supported. Container Insights automatically collects metrics from several workloads, and you can configure it to collect metrics from any workload.

You can adopt Prometheus as an open-source and open-standard method to ingest custom metrics in CloudWatch. The CloudWatch agent with Prometheus support discovers and collects Prometheus metrics to monitor, troubleshoot, and alarm on application performance degradation and failures faster. This also reduces the number of monitoring tools required to improve observability.

Container Insights Prometheus support involves pay-per-use of metrics and logs, including collecting, storing, and analyzing. For more information, see [Amazon CloudWatch Pricing](#).

Pre-built dashboards for some workloads

The Container Insights Prometheus solution includes pre-built dashboards for the popular workloads that are listed in this section. For sample configurations for these workloads, see [\(Optional\) Set up sample containerized Amazon ECS workloads for Prometheus metric testing \(p. 389\)](#) and [\(Optional\) Set up sample containerized Amazon EKS workloads for Prometheus metric testing \(p. 426\)](#).

You can also configure Container Insights to collect Prometheus metrics from other containerized services and applications by editing the agent configuration file.

Workloads with pre-built dashboards for Amazon EKS clusters and Kubernetes clusters running on Amazon EC2 instances:

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy

Workloads with pre-built dashboards for Amazon ECS clusters:

- AWS App Mesh
- Java/JMX
- NGINX
- NGINX Plus

Set up and configure Prometheus metrics collection on Amazon ECS clusters

To collect Prometheus metrics from Amazon ECS clusters, you can use the CloudWatch agent as a collector or use the AWS Distro for OpenTelemetry collector. For information about using the AWS Distro for OpenTelemetry collector, see <https://aws-otel.github.io/docs/getting-started/container-insights/ecs-prometheus>.

The following sections explain how to use the CloudWatch agent as the collector to retrieve Prometheus metrics. You install the CloudWatch agent with Prometheus monitoring on clusters running Amazon ECS, and you can optionally configure the agent to scrape additional targets. These sections also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Container Insights on Amazon ECS supports the following launch type and network mode combinations for Prometheus metrics:

Amazon ECS launch type	Network modes supported
EC2 (Linux)	bridge, host, and awsvpc
Fargate	awsvpc

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters \(p. 376\)](#)
- [Scraping additional Prometheus sources and importing those metrics \(p. 380\)](#)
- [\(Optional\) Set up sample containerized Amazon ECS workloads for Prometheus metric testing \(p. 389\)](#)

Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon ECS. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- Java/JMX

You can also configure the agent to scrape and import metrics from additional Prometheus workloads and sources.

Set up IAM roles

You need two IAM roles for the CloudWatch agent task definition. If you specify `CreateIAMRoles=True` in the AWS CloudFormation stack to have Container Insights create these roles for you, the roles will be created with the correct permissions. If you want to create them yourself or use existing roles, the following roles and permissions are required.

- **CloudWatch agent ECS task role**— The CloudWatch agent container uses this role. It must include the `CloudWatchAgentServerPolicy` policy and a customer-managed policy which contains the following read-only permissions:
 - `ec2:DescribeInstances`
 - `ecs>ListTasks`
 - `ecs>ListServices`
 - `ecs:DescribeContainerInstances`
 - `ecs:DescribeServices`
 - `ecs:DescribeTasks`
 - `ecs:DescribeTaskDefinition`
- **CloudWatch agent ECS task execution role**— This is the role that Amazon ECS requires to launch and execute your containers. Ensure that your task execution role has the `AmazonSSMReadOnlyAccess`, `AmazonECSTaskExecutionRolePolicy`, and `CloudWatchAgentServerPolicy` policies attached. If you want to store more sensitive data for Amazon ECS to use, see [Specifying sensitive data](#).

Install the CloudWatch agent with Prometheus monitoring by using AWS CloudFormation

You use AWS CloudFormation to install the CloudWatch agent with Prometheus monitoring for Amazon ECS clusters. The following list shows the parameters you will use in the AWS CloudFormation template.

- **ECSClusterName**— Specifies the target Amazon ECS cluster.
- **CreateIAMRoles**— Specify `True` to create new roles for the Amazon ECS task role and Amazon ECS task execution role. Specify `False` to reuse existing roles.

- **TaskRoleName**— If you specified **True** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task role. If you specified **False** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task role.
- **ExecutionRoleName**— If you specified **True** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task execution role. If you specified **False** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task execution role.
- **ECSNetworkMode**— If you are using EC2 launch type, specify the network mode here. It must be either **bridge** or **host**.
- **ECSLaunchType**— Specify either **fargate** or **EC2**.
- **SecurityGroupID**— If the **ECSNetworkMode** is **aws-vpc**, specify the security group ID here.
- **SubnetID**— If the **ECSNetworkMode** is **aws-vpc**, specify the subnet ID here.

Command samples

This section includes sample AWS CloudFormation commands to install Container Insights with Prometheus monitoring in various scenarios.

Note

The following setup step pulls the container image from Docker Hub as an anonymous user by default. This pull may be subject to a rate limit. For more information, see [CloudWatch agent container image \(p. 306\)](#).

Create AWS CloudFormation stack for an Amazon ECS cluster in bridge network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=bridge
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/
master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/
cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-
EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
        ParameterKey/CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
        ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
        ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
        ParameterKey=ExecutionRoleName,ParameterValue=${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create AWS CloudFormation stack for an Amazon ECS cluster in host network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=host
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
```

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
        ParameterKey/CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
        ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
        ParameterKey/TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
        ParameterKey/ExecutionRoleName,ParameterValue=${ECS_EXECUTION_ROLE_NAME}
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create AWS CloudFormation stack for an Amazon ECS cluster in awsvpc network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_eks_cluster_name
export ECS_LAUNCH_TYPE=EC2
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_eks_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_eks_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
        ParameterKey/CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
        ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
        ParameterKey=SecurityGroupID,ParameterValue=${ECS_CLUSTER_SECURITY_GROUP}
    \
    ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
    ParameterKey/TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
        ParameterKey/ExecutionRoleName,ParameterValue=${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create AWS CloudFormation stack for a Fargate cluster in awsvpc network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_eks_cluster_name
export ECS_LAUNCH_TYPE=FARGATE
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_eks_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_eks_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-
${ECS_LAUNCH_TYPE}-awsvpc \
--template-body file://cagent-ecs-prometheus-metric-for-awsvpc.yaml \
--parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
ParameterKey/CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
ParameterKey=SecurityGroupID,ParameterValue=${ECS_CLUSTER_SECURITY_GROUP} \
\
ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
ParameterKey=ExecutionRoleName,ParameterValue=${ECS_EXECUTION_ROLE_NAME} \
--capabilities CAPABILITY_NAMED_IAM \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

AWS resources created by the AWS CloudFormation stack

The following table lists the AWS resources that are created when you use AWS CloudFormation to set up Container Insights with Prometheus monitoring on an Amazon ECS cluster.

Resource type	Resource name	Comments
AWS::SSM::Parameter	AmazonCloudWatch-CWAgentConfig-\$ECS_CLUSTER_NAME-\$ECS_LAUNCH_TYPE-\$ECS_NETWORK_MODE	This is the CloudWatch agent with the default App Mesh and Java/JMX embedded metric format definition.
AWS::SSM::Parameter	AmazonCloudWatch-PrometheusConfigName-\$ECS_CLUSTER_NAME-\$ECS_LAUNCH_TYPE-\$ECS_NETWORK_MODE	This is the Prometheus scraping configuration.
AWS::IAM::Role	\$ECS_TASK_ROLE_NAME.	The Amazon ECS task role. This is created only if you specified True for CREATE_IAM_ROLES.
AWS::IAM::Role	\${ECS_EXECUTION_ROLE_NAME}	The Amazon ECS task execution role. This is created only if you specified True for CREATE_IAM_ROLES.
AWS::ECS::TaskDefinition	prometheus-\$ECS_CLUSTER_NAME-\$ECS_LAUNCH_TYPE-\$ECS_NETWORK_MODE	
AWS::ECS::Service	cwagent-prometheus-replica-service-\$ECS_LAUNCH_TYPE-\$ECS_NETWORK_MODE	

Deleting the AWS CloudFormation stack for the CloudWatch agent with Prometheus monitoring

To delete the CloudWatch agent from an Amazon ECS cluster, enter these commands.

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export CLOUDFORMATION_STACK_NAME=your_cloudformation_stack_name
```

```
aws cloudformation delete-stack \  
--stack-name ${CLOUDFORMATION_STACK_NAME} \  
--region ${AWS_DEFAULT_REGION} \  
--profile ${AWS_PROFILE}
```

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in [<scrape_config>](#) in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon ECS clusters, the configurations are integrated with the Parameter Store of AWS Systems Manager by the secrets in the Amazon ECS task definition:

- The secret `PROMETHEUS_CONFIG_CONTENT` is for the Prometheus scrape configuration.
- The secret `CW_CONFIG_CONTENT` is for the CloudWatch agent configuration.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in [<scrape_config>](#) in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:  
  scrape_interval: 1m  
  scrape_timeout: 10s
```

- `scrape_interval`— Defines how frequently to scrape targets.
- `scrape_timeout`— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in the YAML files for Amazon ECS such as `cwagent-ecs-prometheus-metric-for-bridge-host.yaml`, the default scraping jobs are configured in the `ecs_service_discovery` section.

```
"ecs_service_discovery": {  
    "sd_frequency": "1m",
```

```

"sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
"docker_label": {
},
"task_definition_list": [
{
    "sd_job_name": "ecs-appmesh-colors",
    "sd_metrics_ports": "9901",
    "sd_task_definition_arn_pattern": ".*:task-definition\/*-"
ColorTeller-(white):[0-9]+",
    "sd_metrics_path": "/stats/prometheus"
},
{
    "sd_job_name": "ecs-appmesh-gateway",
    "sd_metrics_ports": "9901",
    "sd_task_definition_arn_pattern": ".*:task-definition\/*-"
ColorGateway:[0-9]+",
    "sd_metrics_path": "/stats/prometheus"
}
]
}

```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

Log events from Amazon ECS clusters are stored in the `/aws/ecs/containerinsights/cluster_name/prometheus` log group.

Each scraping job is contained in a different log stream in this log group.

To add a new scraping target, you add a new entry in the `task_definition_list` section under the `ecs_service_discovery` section of the YAML file, and restart the agent. For an example of this process, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics \(p. 422\)](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **`cluster_name`**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon ECS cluster name.
- **`log_group_name`**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses `/aws/ecs/containerinsights/cluster_name/prometheus` for logs from Amazon ECS clusters.
- **`prometheus_config_path`**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **`ecs_service_discovery`**— is the section to specify the configurations of the Amazon ECS Prometheus target auto-discovery functions. Two modes are supported to discover the Prometheus targets: discovery based on the container's docker label or discovery based on the Amazon ECS task definition ARN regular expression. You can use the two modes together and the CloudWatch agent will de-duplicate the discovered targets based on: `{private_ip}:{port}/{metrics_path}`.

The `ecs_service_discovery` section can contain the following fields:

- **`sd_frequency`** is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

This field is optional. The default is 60 seconds (1 minute).

- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for docker label-based service discovery. If you omit this section, docker label-based discovery is not used. This section can contain the following fields:
 - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this docker label, the CloudWatch agent will skip it.
 - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this docker label, the the agent assumes the default path `/metrics`.
 - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
 - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
 - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
 - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
 - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
 - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `service_name_list_for_tasks` is an optional section that you can use to specify the configuration of service name-based discovery. If you omit this section, service name-based discovery is not used. This section can contain the following fields:
 - `sd_service_name_pattern` is the pattern to use to specify the Amazon ECS service where tasks are to be discovered. This is a regular expression.
 - `sd_metrics_ports` Lists the containerPort for the Prometheus metrics. Separate multiple containerPorts with semicolons.
 - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
 - `sd_metrics_path` specifies the Prometheus metrics path. If you omit this, the agent assumes that the default path `/metrics`.
 - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
 - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

If you have multiple labels specified in `source_labels`, we recommend that you do not use ^ or \$ characters in the regular expression for `label_matcher`.

- `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
- `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is ;. You can see this default used in the `label_matcher` line in the following example.
- `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
- `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [
  {
    "source_labels": [ "Service", "Namespace"],
    "label_matcher": "(.*node-exporter.*|.*kube-dns.*);kube-system$",
    "dimensions": [
      ["Service", "Namespace"]
    ],
    "metric_selectors": [
      "^coredns_dns_request_type_count_total$"
    ]
  }
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Name": "coredns_dns_request_type_count_total"
        }
      ],
      "Dimensions": [
        [
          "Namespace",
          "Service"
        ]
      ],
      "Namespace": "ContainerInsights/Prometheus"
    }
  ],
  "Namespace": "kube-system",
  "Service": "kube-dns",
  "coredns_dns_request_type_count_total": 2562,
  "eks_amazonaws_com_component": "kube-dns",
  "instance": "192.168.61.254:9153",
}
```

```
    "job": "kubernetes-service-endpoints",
    ...
}
```

Detailed guide for autodiscovery on Amazon ECS clusters

Prometheus provides dozens of dynamic service-discovery mechanisms as described in [`<scrape_config>`](#). However there is no built-in service discovery for Amazon ECS. The CloudWatch agent adds this mechanism.

When the Amazon ECS Prometheus service discovery is enabled, the CloudWatch agent periodically makes the following API calls to Amazon ECS and Amazon EC2 frontends to retrieve the metadata of the running ECS tasks in the target ECS cluster.

```
EC2:DescribeInstances
ECS>ListTasks
ECS>ListServices
ECS:DescribeContainerInstances
ECS:DescribeServices
ECS:DescribeTasks
ECS:DescribeTaskDefinition
```

The metadata is used by the CloudWatch agent to scan the Prometheus targets within the ECS cluster. The CloudWatch agent supports three service discovery modes:

- Container docker label-based service discovery
- ECS task definition ARN regular expression-based service discovery
- ECS service name regular expression-based service discovery

All modes can be used together. CloudWatch agent de-duplicates the discovered targets based on: `{private_ip}:{port}/{metrics_path}`.

All discovered targets are written into a result file specified by the `sd_result_file` configuration field within the CloudWatch agent container. The following is a sample result file:

```
- targets:
  - 10.6.1.95:32785
labels:
  __metrics_path__: /metrics
  ECS_PROMETHEUS_EXPORTER_PORT: "9406"
  ECS_PROMETHEUS_JOB_NAME: demo-jar-ec2-bridge-dynamic
  ECS_PROMETHEUS_METRICS_PATH: /metrics
  InstanceType: t3.medium
  LaunchType: EC2
  SubnetId: subnet-123456789012
  TaskDefinitionFamily: demo-jar-ec2-bridge-dynamic-port
  TaskGroup: family:demo-jar-ec2-bridge-dynamic-port
  TaskRevision: "7"
  VpcId: vpc-01234567890
  container_name: demo-jar-ec2-bridge-dynamic-port
  job: demo-jar-ec2-bridge-dynamic
- targets:
  - 10.6.3.193:9404
labels:
  __metrics_path__: /metrics
  ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_B: "9404"
  ECS_PROMETHEUS_JOB_NAME: demo-tomcat-ec2-bridge-mapped-port
  ECS_PROMETHEUS_METRICS_PATH: /metrics
  InstanceType: t3.medium
```

```
LaunchType: EC2
SubnetId: subnet-123456789012
TaskDefinitionFamily: demo-tomcat-ec2-bridge-mapped-port
TaskGroup: family:demo-jar-tomcat-bridge-mapped-port
TaskRevision: "12"
VpcId: vpc-01234567890
container_name: demo-tomcat-ec2-bridge-mapped-port
job: demo-tomcat-ec2-bridge-mapped-port
```

You can directly integrate this result file with Prometheus file-based service discovery. For more information about Prometheus file-based service discovery, see [<file_sd_config>](#).

Suppose the result file is written to `/tmp/cwagent_ecs_auto_sd.yaml`. The following Prometheus scrape configuration will consume it.

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: cwagent-ecs-file-sd-config
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/tmp/cwagent_ecs_auto_sd.yaml" ]
```

The CloudWatch agent also adds the following additional labels for the discovered targets.

- `container_name`
- `TaskDefinitionFamily`
- `TaskRevision`
- `TaskGroup`
- `StartedBy`
- `LaunchType`
- `job`
- `__metrics_path__`
- Docker labels

When the cluster has the EC2 launch type, the following three labels are added.

- `InstanceType`
- `VpcId`
- `SubnetId`

Note

Docker labels that don't match the regular expression `[a-zA-Z_][a-zA-Z0-9_]*` are filtered out. This matches the Prometheus conventions as listed in `label_name` in [Configuration file](#) in the Prometheus documentation.

ECS service discovery configuration examples

This section includes examples that demonstrate ECS service discovery.

Example 1

```
"ecs_service_discovery": {
```

```

    "sd_frequency": "1m",
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
    "docker_label": {
    }
}

```

This example enables docker label-based service discovery. The CloudWatch agent will query the ECS tasks' metadata once per minute and write the discovered targets into the /tmp/cwagent_ecs_auto_sd.yaml file within the CloudWatch agent container.

The default value of `sd_port_label` in the `docker_label` section is `ECS_PROMETHEUS_EXPORTER_PORT`. If any running container in the ECS tasks has a `ECS_PROMETHEUS_EXPORTER_PORT` docker label, the CloudWatch agent uses its value as container port to scan all exposed ports of the container. If there is a match, the mapped host port plus the private IP of the container are used to construct the Prometheus exporter target in the following format: `private_ip:host_port`.

The default value of `sd_metrics_path_label` in the `docker_label` section is `ECS_PROMETHEUS_METRICS_PATH`. If the container has this docker label, its value will be used as the `_metrics_path_`. If the container does not have this label, the default value `/metrics` is used.

The default value of `sd_job_name_label` in the `docker_label` section is `job`. If the container has this docker label, its value will be appended as one of the labels for the target to replace the default job name specified in the Prometheus configuration. The value of this docker label is used as the log stream name in the CloudWatch Logs log group.

Example 2

```

"ecs_service_discovery": {
    "sd_frequency": "15s",
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
    "docker_label": {
        "sd_port_label": "ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A",
        "sd_job_name_label": "ECS_PROMETHEUS_JOB_NAME"
    }
}

```

This example enables docker label-based service discovery. The CloudWatch agent will query the ECS tasks' metadata every 15 seconds and write the discovered targets into the /tmp/cwagent_ecs_auto_sd.yaml file within the CloudWatch agent container. The containers with a docker label of `ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A` will be scanned. The value of the docker label `ECS_PROMETHEUS_JOB_NAME` is used as the job name.

Example 3

```

"ecs_service_discovery": {
    "sd_frequency": "5m",
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
    "task_definition_list": [
        {
            "sd_job_name": "java-prometheus",
            "sd_metrics_path": "/metrics",
            "sd_metrics_ports": "9404; 9406",
            "sd_task_definition_arn_pattern": ".*:task-definition/.*javajmx.*:[0-9]+"
        },
        {
            "sd_job_name": "envoy-prometheus",
            "sd_metrics_path": "/stats/prometheus",
            "sd_container_name_pattern": "^envoy$",
            "sd_task_definition_arn_pattern": ".*:task-definition/.*envoy.*:[0-9]+"
        }
    ]
}

```

```

        "sd_metrics_ports": "9901",
        "sd_task_definition_arn_pattern": ".*:task-definition/.*appmesh.*:23"
    }
}

```

This example enables ECS task definition ARN regular expression-based service discovery. The CloudWatch agent will query the ECS tasks' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two task definition ARN regular expression sections are defined:

- For the first section, the ECS tasks with `javajmx` in their ECS task definition ARN are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `_metrics_path_` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, the scraped metrics are sent to the `java-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.
- For the second section, the ECS tasks with `appmesh` in their ECS task definition ARN and with `version` of `:23` are filtered for the container port scan. For containers with a name of `envoy` that expose the container port on 9901, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `_metrics_path_` to `/stats/prometheus`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/prometheus`, and send the scraped metrics to the `envoy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 4

```

"ecs_service_discovery": {
    "sd_frequency": "5m",
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
    "service_name_list_for_tasks": [
        {
            "sd_job_name": "nginx-prometheus",
            "sd_metrics_path": "/metrics",
            "sd_metrics_ports": "9113",
            "sd_service_name_pattern": "^nginx-.*"
        },
        {
            "sd_job_name": "haproxy-prometheus",
            "sd_metrics_path": "/stats/metrics",
            "sd_container_name_pattern": "^haproxy$",
            "sd_metrics_ports": "8404",
            "sd_service_name_pattern": ".*haproxy-service.*"
        }
    ]
}

```

This example enables ECS service name regular expression-based service discovery. The CloudWatch agent will query the ECS services' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two service name regular expression sections are defined:

- For the first section, the ECS tasks that are associated with ECS services that have names matching the regular expression `^nginx-.*` are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9113, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, and the scraped metrics are sent to the `nginx-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.
- or the second section, the ECS tasks that are associated with ECS services that have names matching the regular expression `.*haproxy-service.*` are filtered for the container port scan. For containers with a name of `haproxy` expose the container port on 8404, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/metrics`, and the scraped metrics are sent to the `haproxy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 5

```
"ecs_service_discovery": {
    "sd_frequency": "1m30s",
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
    "docker_label": {
        "sd_port_label": "MY_PROMETHEUS_EXPORTER_PORT_LABEL",
        "sd_metrics_path_label": "MY_PROMETHEUS_METRICS_PATH_LABEL",
        "sd_job_name_label": "MY_PROMETHEUS_METRICS_NAME_LABEL"
    }
    "task_definition_list": [
        {
            "sd_metrics_ports": "9150",
            "sd_task_definition_arn_pattern": "*memcached.*"
        }
    ]
}
```

This example enables both ECS service discovery modes. The CloudWatch agent will query the ECS tasks' metadata every 90 seconds and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

For the docker-based service discovery configuration:

- The ECS tasks with docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` will be filtered for Prometheus port scan. The target Prometheus container port is specified by the value of the label `MY_PROMETHEUS_EXPORTER_PORT_LABEL`.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used for `__metrics_path__`. If the container does not have this docker label, the default value `/metrics` is used.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used as the job label. If the container does not have this docker label, the job name defined in the Prometheus configuration is used.

For the ECS task definition ARN regular expression-based service discovery configuration:

- The ECS tasks with `memcached` in the ECS task definition ARN are filtered for container port scan. The target Prometheus container port is 9150 as defined by `sd_metrics_ports`. The default metrics path `/metrics` is used. The job name defined in the Prometheus configuration is used.

(Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see [Prometheus metrics collected by the CloudWatch agent \(p. 440\)](#).

Topics

- [Sample App Mesh workload for Amazon ECS clusters \(p. 389\)](#)
- [Sample Java/JMX workload for Amazon ECS clusters \(p. 390\)](#)
- [Sample NGINX workload for Amazon ECS clusters \(p. 391\)](#)
- [Sample NGINX Plus workload for Amazon ECS clusters \(p. 396\)](#)
- [Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS \(p. 403\)](#)
- [Tutorial for scraping Redis Prometheus metrics on Amazon ECS Fargate \(p. 408\)](#)

Sample App Mesh workload for Amazon ECS clusters

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS \(p. 307\)](#).

First, follow this [walkthrough](#) to deploy the sample color app on your Amazon ECS cluster. After you finish, you will have App Mesh Prometheus metrics exposed on port 9901.

Next, follow these steps to install the CloudWatch agent with Prometheus monitoring on the same Amazon ECS cluster where you installed the color app. The steps in this section install the CloudWatch agent in bridge network mode.

The environment variables `ENVIRONMENT_NAME`, `AWS_PROFILE`, and `AWS_DEFAULT_REGION` that you set in the walkthrough will also be used in the following steps.

To install the CloudWatch agent with Prometheus monitoring for testing

1. Download the AWS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Set the network mode by entering the following commands.

```
export ECS_CLUSTER_NAME=${ENVIRONMENT_NAME}
export ECS_NETWORK_MODE=bridge
```

3. Create the AWS CloudFormation stack by entering the following commands.

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS- ${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
        ParameterKey=CreateIAMRoles,ParameterValue=True \
        ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
        ParameterKey=TaskRoleName,ParameterValue=CWAgent-Prometheus-TaskRole- ${ECS_CLUSTER_NAME} \
```

```
ParameterKey=ExecutionRoleName,ParameterValue=CWAgent-Prometheus-ExecutionRole-`${ECS_CLUSTER_NAME} \
--capabilities CAPABILITY_NAMED_IAM \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

4. (Optional) When the AWS CloudFormation stack is created, you see a CREATE_COMPLETE message. If you want to check the status before you see that message, enter the following command.

```
aws cloudformation describe-stacks \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--query 'Stacks[0].StackStatus' \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Troubleshooting

The steps in the walkthrough use jq to parse the output result of the AWS CLI. For more information about installing jq, see [jq](#). Use the following command to set the default output format of your AWS CLI to JSON so jq can parse it correctly.

```
$ aws configure
```

When the response gets to Default output format, enter **json**.

Uninstall the CloudWatch agent with Prometheus monitoring

When you are finished testing, enter the following command to uninstall the CloudWatch agent by deleting the AWS CloudFormation stack.

```
aws cloudformation delete-stack \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Sample Java/JMX workload for Amazon ECS clusters

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see [prometheus/jmx_exporter](#).

The CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the JMX Exporter to expose the metrics on a different port or metrics_path. If you do change the port or path, update the default ecs_service_discovery section in the CloudWatch agent configuration.

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS \(p. 307\)](#).

To install the Java/JMX sample workload for Amazon ECS clusters

1. Follow the steps in these sections to create your Docker images.
 - [Example: Java Jar Application Docker image with Prometheus metrics \(p. 434\)](#)
 - [Example: Apache Tomcat Docker image with Prometheus metrics \(p. 432\)](#)

2. Specify the following two docker labels in the Amazon ECS task definition file. You can then run the task definition as an Amazon ECS service or Amazon ECS task in the cluster.
 - Set `ECS_PROMETHEUS_EXPORTER_PORT` to point to the containerPort where the Prometheus metrics are exposed.
 - Set `Java_EMF_Metrics` to `true`. The CloudWatch agent uses this flag to generate the embedded metric format in the log event.

The following is an example:

```
{
  "family": "workload-java-ec2-bridge",
  "taskRoleArn": "{{task-role-arn}}",
  "executionRoleArn": "{{execution-role-arn}}",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "tomcat-prometheus-workload-java-ec2-bridge-dynamic-port",
      "image": "your_docker_image_tag_for_tomcat_with_prometheus_metrics",
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 9404
        }
      ],
      "dockerLabels": {
        "ECS_PROMETHEUS_EXPORTER_PORT": "9404",
        "Java_EMF_Metrics": "true"
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "256",
  "memory": "512"
}
```

The default setting of the CloudWatch agent in the AWS CloudFormation template enables both docker label-based service discovery and task definition ARN-based service discovery. To view these default settings, see line 65 of the [CloudWatch agent YAML configuration file](#). The containers with the `ECS_PROMETHEUS_EXPORTER_PORT` label will be auto-discovered based on the specified container port for Prometheus scraping.

The default setting of the CloudWatch agent also has the `metric_declarator` setting for Java/JMX at line 112 of the same file. All docker labels of the target containers will be added as additional labels in the Prometheus metrics and sent to CloudWatch Logs. For the Java/JMX containers with docker label `Java_EMF_Metrics="true"`, the embedded metric format will be generated.

Sample NGINX workload for Amazon ECS clusters

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see [nginx-prometheus-exporter](#) on Github. For more information about the NGINX reverse proxy, see [ecs-nginx-reverse-proxy](#) on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus

Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy>.
2. Find the `app` directory and build an image from that directory:

```
docker build -t web-server-app ./path-to-app-directory
```

3. Build a custom image for NGINX. First, create a directory with the following two files:

- A sample Dockerfile:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

- An `nginx.conf` file, modified from <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/>:

```
events {
    worker_connections 768;
}

http {
    # Nginx will handle gzip compression of responses from the app server
    gzip on;
    gzip_proxied any;
    gzip_types text/plain application/json;
    gzip_min_length 1000;

    server{
        listen 8080;
        location /stub_status {
            stub_status on;
        }
    }

    server {
        listen 80;

        # Nginx will reject anything not matching /api
        location /api {
            # Reject requests with unsupported HTTP method
            if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
                return 405;
            }

            # Only requests matching the whitelist expectations will
            # get sent to the application server
            proxy_pass http://app:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
        }
    }
}
```

```
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_cache_bypass $http_upgrade;
}
}
```

Note

`stub_status` must be enabled on the same port that `nginx-prometheus-exporter` is configured to scrape metrics from. In our example task definition, `nginx-prometheus-exporter` is configured to scrape metrics from port 8080.

- #### 4. Build an image from files in your new directory:

```
docker build -t nginx-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`. The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace `your-customized-nginx-image` with the image URI for your customized NGINX image, and replace `your-web-server-app-image` with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "links": [
        "app"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 256,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
```

```

    "cpu": 256,
    "essential": true,
    "command": [
        "-nginxx.scrape-uri",
        "http://nginx:8080/stub_status"
    ],
    "links": [
        "nginx"
    ],
    "portMappings": [
        {
            "containerPort": 9113,
            "protocol": "tcp"
        }
    ]
},
"networkMode": "bridge",
"placementConstraints": [],
"family": "nginx-sample-stack"
}

```

2. Register the task definition by entering the following command.

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition.json
```

3. Create a service to run the task by entering the following command:

Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-service$`. The next section provides more details.

```
aws ecs create-service \
--cluster your-cluster-name \
--service-name nginx-service \
--task-definition nginx-sample-stack:1 \
--desired-count 1
```

Configure the CloudWatch agent to scrape NGINX Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task via the service name pattern, and the port 9113, where the exporter exposes the Prometheus metrics for NGINX. With the task discovered and the metrics available, the CloudWatch agent begins posting the collected metrics to the log stream `nginx-prometheus-exporter`.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```

"service_name_list_for_tasks": [
    {
        "sd_job_name": "nginx-prometheus-exporter",
        "sd_metrics_path": "/metrics",
        "sd_metrics_ports": "9113",
        "sd_service_name_pattern": "^nginx-service$"
    }
],

```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX metrics. Be sure to follow the existing indentation pattern.

```

{
    "source_labels": ["job"],
    "label_matcher": ".*nginx.*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
    "metric_selectors": [
        "^(nginx_.*$"
    ]
},

```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using AWS CloudFormation, you can create a change set by entering the following commands:

```

ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
        ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
        ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
        ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
        ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME \
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-scraping-support

```

5. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
6. Review the newly-created changeset **nginx-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restart the CloudWatch agent task by entering the following command:

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION

```

7. Wait about 10 seconds, and then enter the following command.

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \

```

```
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
        ParameterKey/CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
        ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
        ParameterKey/TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
        ParameterKey/ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your NGINX metrics and logs

You can now view the NGINX metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/your_cluster_name/prometheus**, in the log stream **nginx-prometheus-exporter**.

Sample NGINX Plus workload for Amazon ECS clusters

NGINX Plus is the commerical version of NGINX. You must have a licence to use it. For more information, see [NGINX Plus](#)

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX Plus reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see [nginx-prometheus-exporter](#) on Github. For more information about the NGINX reverse proxy, see [ecs-nginx-reverse-proxy](#) on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Plus Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX Plus reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX Plus reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy>.
2. Find the app directory and build an image from that directory:

```
docker build -t web-server-app ./path-to-app-directory
```

3. Build a custom image for NGINX Plus. Before you can build the image for NGINX Plus, you need to obtain the key named `nginx-repo.key` and the SSL certificate `nginx-repo.crt` for your licensed NGINX Plus. Create a directory and store in it your `nginx-repo.key` and `nginx-repo.crt` files.

In the directory that you just created, create the following two files:

- A sample Dockerfile with the following content. This docker file is adopted from a sample file provided at https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/#docker_plus_image. The important change that we make is that we load a separate file, called `nginx.conf`, which will be created in the next step.

```
FROM debian:buster-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>

# Define NGINX versions for NGINX Plus and NGINX Plus modules
# Uncomment this block and the versioned nginxPackages block in the main RUN
# instruction to install a specific release
# ENV NGINX_VERSION 21
# ENV NJS_VERSION 0.3.9
# ENV PKG_RELEASE 1~buster

# Download certificate and key from the customer portal (https://cs.nginx.com
# (https://cs.nginx.com/))
# and copy to the build context
COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/
# COPY nginx.conf /etc/ssl/nginx/nginx.conf

RUN set -x \
    # Create nginx user/group first, to be consistent throughout Docker variants
    && addgroup --system --gid 101 nginx \
    && adduser --system --disabled-login --ingroup nginx --no-create-home --home /
    nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
    && apt-get update \
    && apt-get install --no-install-recommends --no-install-suggests -y ca-certificates
        gnupg1 \
    && \
    NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
    found=''; \
    for server in \
        ha.pool.sks-keyservers.net (http://ha.pool.sks-keyservers.net/) \
        hkp://keyserver.ubuntu.com:80 \
        hkp://p80.pool.sks-keyservers.net:80 \
        pgp.mit.edu (http://pgp.mit.edu/) \
    ; do \
        echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
        apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys
            "$NGINX_GPGKEY" && found=yes && break; \
    done; \
    test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" && exit
    1; \
```

```

apt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \
# Install the latest release of NGINX Plus and/or NGINX Plus modules
# Uncomment individual modules if necessary
# Use versioned packages over defaults to specify a release
&& nginxPackages=" \
nginx-plus \
# nginx-plus=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-xslt \
# nginx-plus-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-geoip \
# nginx-plus-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-image-filter \
# nginx-plus-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-perl \
# nginx-plus-module-perl=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-njs \
# nginx-plus-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
" \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslCert \"/etc/ssl/nginx/nginx-repo.crt
\";" >> /etc/apt/apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslKey \"/etc/ssl/nginx/nginx-repo.key
\";" >> /etc/apt/apt.conf.d/90nginx \
&& printf "deb https://plus-pkgs.nginx.com/debian buster nginx-plus\n" > /etc/apt/
sources.list.d/nginx-plus.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
$nginxPackages \
gettext-base \
curl \
&& apt-get remove --purge --auto-remove -y && rm -rf /var/lib/apt/lists/* /etc/apt/
sources.list.d/nginx-plus.list \
&& rm -rf /etc/apt/apt.conf.d/90nginx /etc/ssl/nginx

# Forward request logs to Docker log collector
RUN in -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

STOP SIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]

```

- An `nginx.conf` file, modified from <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/nginx>.

```

events {
    worker_connections 768;
}

http {
    # Nginx will handle gzip compression of responses from the app server
    gzip on;
    gzip_proxied any;
    gzip_types text/plain application/json;
    gzip_min_length 1000;

    upstream backend {
        zone name 10m;

```

```

        server app:3000    weight=2;
        server app2:3000   weight=1;
    }

    server{
        listen 8080;
        location /api {
            api write=on;
        }
    }

    match server_ok {
        status 100-599;
    }

    server {
        listen 80;
        status_zone zone;
        # Nginx will reject anything not matching /api
        location /api {
            # Reject requests with unsupported HTTP method
            if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
                return 405;
            }

            # Only requests matching the whitelist expectations will
            # get sent to the application server
            proxy_pass http://backend;
            health_check uri=/lorem-ipsum match=server_ok;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_cache_bypass $http_upgrade;
        }
    }
}
}

```

4. Build an image from files in your new directory:

```
docker build -t nginx-plus-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX Plus and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Plus Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`. The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace `your-customized-nginx-plus-image` with the image URI for your customized NGINX Plus image, and replace `your-web-server-app-image` with the image URI for your web server app image.

```
{
    "containerDefinitions": [

```

```
{
    "name": "nginx",
    "image": "your-customized-nginx-plus-image",
    "memory": 256,
    "cpu": 256,
    "essential": true,
    "portMappings": [
        {
            "containerPort": 80,
            "protocol": "tcp"
        }
    ],
    "links": [
        "app",
        "app2"
    ]
},
{
    "name": "app",
    "image": "your-web-server-app-image",
    "memory": 256,
    "cpu": 128,
    "essential": true
},
{
    "name": "app2",
    "image": "your-web-server-app-image",
    "memory": 256,
    "cpu": 128,
    "essential": true
},
{
    "name": "nginx-prometheus-exporter",
    "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
    "memory": 256,
    "cpu": 256,
    "essential": true,
    "command": [
        "-nginx.plus",
        "-nginx.scrape-uri",
        "http://nginx:8080/api"
    ],
    "links": [
        "nginx"
    ],
    "portMappings": [
        {
            "containerPort": 9113,
            "protocol": "tcp"
        }
    ]
},
{
    "networkMode": "bridge",
    "placementConstraints": [],
    "family": "nginx-plus-sample-stack"
}
```

2. Register the task definition:

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition.json
```

3. Create a service to run the task by entering the following command:

```
aws ecs create-service \
--cluster your-cluster-name \
--service-name nginx-plus-service \
--task-definition nginx-plus-sample-stack:1 \
--desired-count 1
```

Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-plus-service$`. The next section provides more details.

Configure the CloudWatch agent to scrape NGINX Plus Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task via the service name pattern, and the port 9113, where the exporter exposes the prometheus metrics for NGINX. With the task discovered and the metrics available, the CloudWatch agent begins posting the collected metrics to the log stream `nginx-prometheus-exporter`.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [
{
    "sd_job_name": "nginx-plus-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-plus.*"
}
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX Plus metrics. Be sure to follow the existing indentation pattern.

```
{
    "source_labels": ["job"],
    "label_matcher": "^nginx-plus.*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
    "metric_selectors": [
        "^nginxplus_connections_accepted$",
        "^nginxplus_connections_active$",
        "^nginxplus_connections_dropped$",
        "^nginxplus_connections_idle$",
        "^nginxplus_http_requests_total$",
        "^nginxplus_ssl_handshakes$",
        "^nginxplus_ssl_handshakes_failed$",
        "^nginxplus_up$",
        "^nginxplus_upstream_server_health_checks_fails$"
    ]
}
```

```

        ],
},
{
    "source_labels": ["job"],
    "label_matcher": "^nginx-plus.*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "upstream"]],
    "metric_selectors": [
        "^(nginxplus_upstream_server_response_time$"
    ]
},
{
    "source_labels": ["job"],
    "label_matcher": "^nginx-plus.*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "code"]],
    "metric_selectors": [
        "^(nginxplus_upstream_server_responses$",
        "^(nginxplus_server_zone_responses$"
    ]
}
,
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using AWS CloudFormation, you can create a change set by entering the following commands:

```

ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
        ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
        ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
        ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
        ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-plus-scraping-support

```

5. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
6. Review the newly-created changeset **nginx-plus-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restart the CloudWatch agent task by entering the following command:

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION

```

7. Wait about 10 seconds, and then enter the following command.

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION

```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```

ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
        ParameterKey/CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
        ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
        ParameterKey/TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
        ParameterKey/ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME \
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
  
```

Viewing your NGINX Plus metrics and logs

You can now view the NGINX Plus metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/your_cluster_name/prometheus**, in the log stream **nginx-plus-prometheus-exporter**.

Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Memcached application on an Amazon Amazon ECS cluster with the EC2 launch type. The Memcached Prometheus exporter target will be auto-discovered by the CloudWatch agent by ECS task definition-based service discovery.

Memcached is a general-purpose distributed memory caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read. For more information, see [What is Memcached?](#)

The [memcached_exporter](#) (Apache License 2.0) is one of the Prometheus official exporters. By default the memcache_exporter serves on port 0.0.0.0:9150 at /metrics.

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [Memcached](#)
- [prom/memcached-exporter](#)

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS \(p. 307\)](#).

Topics

- [Set the Amazon ECS EC2 cluster environment variables \(p. 404\)](#)
- [Install the sample Memcached workload \(p. 404\)](#)
- [Configure the CloudWatch agent to scrape Memcached Prometheus metrics \(p. 405\)](#)
- [Viewing your Memcached metrics \(p. 407\)](#)

Set the Amazon ECS EC2 cluster environment variables

To set the Amazon ECS EC2 cluster environment variables

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see [Installing the Amazon ECS CLI](#).
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-ec2-memcached-tutorial  
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS cluster with the EC2 launch type where you want to install the sample Memcached workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam --size 1 \  
--instance-type t3.medium \  
--cluster $ECS_CLUSTER_NAME \  
--region $AWS_REGION
```

The expected result of this command is as follows:

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.  
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.44.4 and Docker  
version 19.03.6-ce  
INFO[0001] Created cluster cluster=ecs-ec2-memcached-  
tutorial region=ca-central-1  
INFO[0002] Waiting for your cluster resources to be created...  
INFO[0002] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS  
INFO[0063] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS  
INFO[0124] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS  
VPC created: vpc-xxxxxxxxxxxxxxxxxxxx  
Security Group created: sg-xxxxxxxxxxxxxxxxxxxx  
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxx  
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxx  
Cluster creation succeeded.
```

Install the sample Memcached workload

To install the sample Memcached workload which exposes the Prometheus metrics

1. Download the Memcached AWS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/sample_traffic/memcached/memcached-traffic-sample.yaml
```

2. Set the IAM role names to be created for Memcached by entering the following commands.

```
MEMCACHED_ECS_TASK_ROLE_NAME=memcached-prometheus-demo-ecs-task-role-name  
MEMCACHED_ECS_EXECUTION_ROLE_NAME=memcached-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Memcached workload by entering the following command. This sample installs the workload in host network mode.

```
MEMCACHED_ECS_NETWORK_MODE=host

aws cloudformation create-stack --stack-name Memcached-Prometheus-Demo-ECS-  
$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MODE \  
    --template-body file://memcached-traffic-sample.yaml \  
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \  
        ParameterKey=ECSNetworkMode,ParameterValue=$MEMCACHED_ECS_NETWORK_MODE  
    \  
        ParameterKey=TaskRoleName,ParameterValue=$MEMCACHED_ECS_TASK_ROLE_NAME  
    \  
        ParameterKey=ExecutionRoleName,ParameterValue=  
$MEMCACHED_ECS_EXECUTION_ROLE_NAME \  
    --capabilities CAPABILITY_NAMED_IAM \  
    --region $AWS_REGION
```

The AWS CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Memcached task definition
- One Memcached service

In the Memcached task definition, two containers are defined:

- The primary container runs a simple Memcached application and opens port 11211 for access.
- The other container runs the Redis exporter process to expose the Prometheus metrics on port 9150. This is the container to be discovered and scraped by the CloudWatch agent.

Configure the CloudWatch agent to scrape Memcached Prometheus metrics

To configure the CloudWatch agent to scrape Memcached Prometheus metrics

1. Download the latest version of cwagent-ecs-prometheus-metric-for-awsvpc.yaml by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the value key in the resource:CWAgentConfigSSMParameter section.

Then, in the `ecs_service_discovery` section, add the following configuration into the `task_definition_list` section.

```
{
    "sd_job_name": "ecs-memcached",
    "sd_metrics_ports": "9150",
    "sd_task_definition_arn_pattern": ".*:task-definition/memcached-prometheus-demo.*:[0-9]+"
},
```

For the `metric_declaration` section, the default setting does not allow any Memcached metrics. Add the following section to allow Memcached metrics. Be sure to follow the existing indentation pattern.

```
{
    "source_labels": ["container_name"],
    "label_matcher": "memcached-exporter-*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily"]],
    "metric_selectors": [
        "^memcached_current_(bytes|items|connections)$",
        "^memcached_items_(reclaimed|evicted)_total$",
        "^memcached_(written|read)_bytes_total$",
        "^memcached_limit_bytes$",
        "^memcached_commands_total$"
    ],
},
{
    "source_labels": ["container_name"],
    "label_matcher": "memcached-exporter-*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "status", "command"], ["ClusterName", "TaskDefinitionFamily", "command"]],
    "metric_selectors": [
        "^memcached_commands_total$"
    ]
},
```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by AWS CloudFormation, you can create a change set by entering the following commands.

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
        ParameterKey/CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
        ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
        ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
        ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME \
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name memcached-scraping-support
```

4. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
5. Review the newly created changeset `memcached-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-$ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
        ParameterKey/CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
        ParameterKey/ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
        ParameterKey/TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
        ParameterKey/ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME \
    \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your Memcached metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions
memcached_current_items	ClusterName, TaskDefinitionFamily
memcached_current_connections	ClusterName, TaskDefinitionFamily
memcached_limit_bytes	ClusterName, TaskDefinitionFamily
memcached_current_bytes	ClusterName, TaskDefinitionFamily
memcached_written_bytes	ClusterName, TaskDefinitionFamily
memcached_read_bytes	ClusterName, TaskDefinitionFamily
memcached_items_evicted	ClusterName, TaskDefinitionFamily
memcached_items_recycled	ClusterName, TaskDefinitionFamily
memcached_commands_total	ClusterName, TaskDefinitionFamily, command

Metric name	Dimensions	
	ClusterName, TaskDefinitionFamily, status, command	

Note

The value of the **command** dimension can be: delete, get, cas, set, decr, incr, or flush.

The value of the **status** dimension can be hit, miss, or badval.

You can also create a CloudWatch dashboard for your Memcached Prometheus metrics.

To create a dashboard for Memcached Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_memcached_cw_dashboard_name
ECS_TASK_DEF_FAMILY=memcached-prometheus-demo-$ECS_CLUSTER_NAME-EC2-
$MEMCACHED_ECS_NETWORK_MOD
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/sample_cloudwatch_dashboards/memcached/cw_dashboard_memcached.json \
| sed "s/{{YOUR_AWS_REGION}}/$AWS_REGION/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/$ECS_CLUSTER_NAME/g" \
| sed "s/{{YOUR_TASK_DEF_FAMILY}}/$ECS_TASK_DEF_FAMILY/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --region
$AWS_REGION --dashboard-body
```

Tutorial for scraping Redis Prometheus metrics on Amazon ECS Fargate

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis application in an Amazon ECS Fargate cluster. The Redis Prometheus exporter target will be auto-discovered by the CloudWatch agent with Prometheus metric support based on the container's docker labels.

Redis (<https://redis.io/>) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see [redis](#).

`redis_exporter` (MIT License licensed) is used to expose the Redis prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see [redis_exporter](#).

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [redis](#)
- [redis_exporter](#)

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS \(p. 307\)](#).

Topics

- [Set the Amazon ECS Fargate cluster environment variable \(p. 409\)](#)

- Set the network environment variables for the Amazon ECS Fargate cluster (p. 409)
- Install the sample Redis workload (p. 410)
- Configure the CloudWatch agent to scrape Redis Prometheus metrics (p. 411)
- Viewing your Redis metrics (p. 413)

Set the Amazon ECS Fargate cluster environment variable

To set the Amazon ECS Fargate cluster environment variable

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see [Installing the Amazon ECS CLI](#).
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-fargate-redis-tutorial  
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS Fargate cluster where you want to install the sample Redis workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam \  
--cluster $ECS_CLUSTER_NAME \  
--launch-type FARGATE \  
--region $AWS_DEFAULT_REGION
```

The expected result of this command is as follows:

```
INFO[0000] Created cluster    cluster=ecs-fargate-redis-tutorial region=ca-central-1  
INFO[0001] Waiting for your cluster resources to be created...  
INFO[0001] Cloudformation stack status    stackStatus=CREATE_IN_PROGRESS  
VPC created: vpc-xxxxxxxxxxxxxxxxxxxx  
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxx  
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxx  
Cluster creation succeeded.
```

Set the network environment variables for the Amazon ECS Fargate cluster

To set the network environment variables for the Amazon ECS Fargate cluster

1. Set your VPC and subnet ID of the Amazon ECS cluster. If you created a new cluster in the previous procedure, you'll see these values in the result of the final command. Otherwise, use the IDs of the existing cluster that you are going to use with Redis.

```
ECS_CLUSTER_VPC=vpc-xxxxxxxxxxxxxxxxxxxx  
ECS_CLUSTER_SUBNET_1=subnet-xxxxxxxxxxxxxxxxxxxx  
ECS_CLUSTER_SUBNET_2=subnet-xxxxxxxxxxxxxxxxxxxx
```

2. In this tutorial, we are going to install the Redis application and the CloudWatch agent in the default security group of the Amazon ECS cluster's VPC. The default security group allows all network connection within the same security group so the CloudWatch agent can scrape the Prometheus metrics exposed on the Redis containers. In a real production environment, you might want to create dedicated security groups for the Redis application and CloudWatch agent and set customized permissions for them.

Enter the following command to get the default security group ID.

```
aws ec2 describe-security-groups \
--filters Name=vpc-id,Values=$ECS_CLUSTER_VPC \
--region $AWS_DEFAULT_REGION
```

Then set the Fargate cluster default security group variable by entering the following command, replacing *my-default-security-group* with the value you found from the previous command.

```
ECS_CLUSTER_SECURITY_GROUP=my-default-security-group
```

Install the sample Redis workload

To install the sample Redis workload which exposes the Prometheus metrics

1. Download the Redis AWS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml
```

2. Set the IAM role names to be created for Redis by entering the following commands.

```
REDIS_ECS_TASK_ROLE_NAME=redis-prometheus-demo-ecs-task-role-name
REDIS_ECS_EXECUTION_ROLE_NAME=redis-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Redis workload by entering the following command.

```
aws cloudformation create-stack --stack-name Redis-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME=fargate-awsvpc \
--template-body file://redis-traffic-sample.yaml \
--parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET_1 \
ParameterKey=TaskRoleName,ParameterValue=$REDIS_ECS_TASK_ROLE_NAME \
ParameterKey=ExecutionRoleName,ParameterValue=
$REDIS_ECS_EXECUTION_ROLE_NAME \
--capabilities CAPABILITY_NAMED_IAM \
--region $AWS_DEFAULT_REGION
```

The AWS CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Redis task definition
- One Redis service

In the Redis task definition, two containers are defined:

- The primary container runs a simple Redis application and opens port 6379 for access.
- The other container runs the Redis exporter process to expose the Prometheus metrics on port 9121. This is the container to be discovered and scraped by the CloudWatch agent. The following docker label is defined so that the CloudWatch agent can discover this container based on it.

```
ECS_PROMETHEUS_EXPORTER_PORT: 9121
```

Configure the CloudWatch agent to scrape Redis Prometheus metrics

To configure the CloudWatch agent to scrape Redis Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

Then, in the `ecs_service_discovery` section shown here, the `docker_label`-based service discovery is enabled with the default settings which are based on `ECS_PROMETHEUS_EXPORTER_PORT`, which matches the docker label we defined in the Redis ECS task definition. So we do not need to make any changes in this section:

```
ecs_service_discovery": {  
    "sd_frequency": "1m",  
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",  
    * "docker_label": {  
        },*  
    ...
```

For the `metric_declaration` section, the default setting does not allow any Redis metrics. Add the following section to allow Redis metrics. Be sure to follow the existing indentation pattern.

```
{  
    "source_labels": ["container_name"],  
    "label_matcher": "^redis-exporter-.*$",  
    "dimensions": [["ClusterName", "TaskDefinitionFamily"]],  
    "metric_selectors": [  
        "^redis_net_(in|out)put_bytes_total$"  
        "^redis_(expired|evicted)_keys_total$"  
        "^redis_keyspace_(hits|misses)_total$"  
        "^redis_memory_used_bytes$"  
        "^redis_connected_clients$"  
    ]  
},  
{  
    "source_labels": ["container_name"],  
    "label_matcher": "^redis-exporter-.*$",  
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "cmd"]],  
    "metric_selectors": [  
        "^redis_commands_total$"  
    ]  
},  
{  
    "source_labels": ["container_name"],  
    "label_matcher": "^redis-exporter-.*$",  
    "dimensions": [["ClusterName", "TaskDefinitionFamily", "db"]],  
    "metric_selectors": [  
        "^redis_db_keys$"  
    ]  
}
```

},

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by AWS CloudFormation, you can create a change set by entering the following commands.

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
--template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
--parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
ParameterKey/CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
\
--capabilities CAPABILITY_NAMED_IAM \
--region ${AWS_DEFAULT_REGION} \
--change-set-name redis-scraping-support
```

4. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
5. Review the newly created changeset `redis-scraping-support`. You should see one change applied to the `cwAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-$ECS_CLUSTER_NAME-
$ECS_LAUNCH_TYPE-awsvpc \
--template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
--parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
ParameterKey/CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
```

```

ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
\ 
--capabilities CAPABILITY_NAMED_IAM \
--region ${AWS_DEFAULT_REGION}

```

Viewing your Redis metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric Name	Dimensions
redis_net_input_bytes	ClusterName, TaskDefinitionFamily
redis_net_output_bytes	ClusterName, TaskDefinitionFamily
redis_expired_keys_total	ClusterName, TaskDefinitionFamily
redis_evicted_keys_total	ClusterName, TaskDefinitionFamily
redis_keyspace_hits_total	ClusterName, TaskDefinitionFamily
redis_keyspace misses_total	ClusterName, TaskDefinitionFamily
redis_memory_used_by	ClusterName, TaskDefinitionFamily
redis_connected_clients	ClusterName, TaskDefinitionFamily
redis_commands_total	ClusterName, TaskDefinitionFamily, cmd
redis_db_keys	ClusterName, TaskDefinitionFamily, db

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.
The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis Prometheus metrics.

To create a dashboard for Redis Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```

DASHBOARD_NAME=your_cw_dashboard_name
ECS_TASK_DEF_FAMILY=redis-prometheus-demo-$ECS_CLUSTER_NAME-fargate-awsvpc

```

2. Enter the following command to create the dashboard.

```

curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/master/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \

```

Set up and configure Prometheus metrics collection on Amazon EKS and Kubernetes clusters

To collect Prometheus metrics from clusters running Amazon EKS or Kubernetes, you can use the CloudWatch agent as a collector or use the AWS Distro for OpenTelemetry collector. For information about using the AWS Distro for OpenTelemetry collector, see <https://aws-otel.github.io/docs/getting-started/container-insights/eks-prometheus>.

The following sections explain how to collect Prometheus metrics using the CloudWatch agent. They explain how to install the CloudWatch agent with Prometheus monitoring on clusters running Amazon EKS or Kubernetes, and how to configure the agent to scrape additional targets. They also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Topics

- [Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters \(p. 414\)](#)
- [Prometheus metric type conversion by the CloudWatch Agent \(p. 439\)](#)
- [Prometheus metrics collected by the CloudWatch agent \(p. 440\)](#)
- [Viewing your Prometheus metrics \(p. 447\)](#)
- [Prometheus metrics troubleshooting \(p. 448\)](#)

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters \(p. 415\)](#)
- [Scraping additional Prometheus sources and importing those metrics \(p. 418\)](#)

- [\(Optional\) Set up sample containerized Amazon EKS workloads for Prometheus metric testing \(p. 426\)](#)

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Setting up IAM roles \(p. 415\)](#)
- [Installing the CloudWatch agent to collect Prometheus metrics \(p. 416\)](#)

Setting up IAM roles

The first step is to set up the necessary IAM role in the cluster. There are two methods:

- Set up an IAM role for a service account, also known as a *service role*. This method works for both the EC2 launch type and the Fargate launch type.
- Add an IAM policy to the IAM role used for the cluster. This works only for the EC2 launch type.

Set up a service role (EC2 launch type and Fargate launch type)

To set up a service role, enter the following command. Replace *MyCluster* with the name of the cluster.

```
eksctl create iamserviceaccount \
--name cwagent-prometheus \
--namespace amazon-cloudwatch \
--cluster MyCluster \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

Add a policy to the cluster's IAM role (EC2 launch type only)

To set up the IAM policy in a cluster for Prometheus support

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, choose **Instances**.
3. You need to find the prefix of the IAM role name for the cluster. To do this, select the check box next to the name of an instance that is in the cluster, and choose **Actions**, **Instance Settings**, **Attach/Replace IAM Role**. Then copy the prefix of the IAM role, such as eksctl-dev303-workshop-nodegroup.
4. Open the IAM console at <https://console.aws.amazon.com/iam/>.
5. In the navigation pane, choose **Roles**.
6. Use the search box to find the prefix that you copied earlier in this procedure, and choose that role.
7. Choose **Attach policies**.
8. Use the search box to find **CloudWatchAgentServerPolicy**. Select the check box next to **CloudWatchAgentServerPolicy**, and choose **Attach policy**.

Installing the CloudWatch agent to collect Prometheus metrics

You must install the CloudWatch agent in the cluster to collect the metrics. How to install the agent differs for Amazon EKS clusters and Kubernetes clusters.

Delete previous versions of the CloudWatch agent with Prometheus support

If you have already installed a version of the CloudWatch agent with Prometheus support in your cluster, you must delete that version by entering the following command. This is necessary only for previous versions of the agent with Prometheus support. You do not need to delete the CloudWatch agent that enables Container Insights without Prometheus support.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

Installing the CloudWatch agent on Amazon EKS clusters with the EC2 launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster

1. Enter the following command to check whether the `amazon-cloudwatch` namespace has already been created:

```
kubectl get namespace
```

2. If `amazon-cloudwatch` is not displayed in the results, create it by entering the following command:

```
kubectl create namespace amazon-cloudwatch
```

3. To deploy the agent with the default configuration and have it send data to the AWS Region that it is installed in, enter the following command:

Note

The following setup step pulls the container image from Docker Hub as an anonymous user by default. This pull may be subject to a rate limit. For more information, see [CloudWatch agent container image \(p. 306\)](#).

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

To have the agent send data to a different Region instead, follow these steps:

- a. Download the YAML file for the agent by entering the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

- b. Open the file with a text editor, and search for the `cwagentconfig.json` block of the file.
- c. Add the highlighted lines, specifying the Region that you want:

```
cwagentconfig.json: |
{
  "agent": {
    "region": "us-east-2"
  },
  "logs": { ... }
```

- d. Save the file and deploy the agent using your updated file.

```
kubectl apply -f prometheus-eks.yaml
```

Installing the CloudWatch agent on Amazon EKS clusters with the Fargate launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type

1. Enter the following command to create a Fargate profile for the CloudWatch agent so that it can run inside the cluster. Replace `MyCluster` with the name of the cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--name amazon-cloudwatch \
--namespace amazon-cloudwatch
```

2. To install the CloudWatch agent, enter the following command. Replace `MyCluster` with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

Replace `region` with the name of the Region where you want the metrics to be sent. For example, `us-west-1`.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml |
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
kubectl apply -f -
```

Installing the CloudWatch agent on a Kubernetes cluster

To install the CloudWatch agent with Prometheus support on a cluster running Kubernetes, enter the following command:

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml | sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" | kubectl apply -f -
```

Replace **MyCluster** with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

Replace **region** with the name of the AWS Region where you want the metrics to be sent. For example, **us-west-1**.

Verify that the agent is running

On both Amazon EKS and Kubernetes clusters, you can enter the following command to confirm that the agent is running.

```
kubectl get pod -l "app=cwagent-prometheus" -n amazon-cloudwatch
```

If the results include a single CloudWatch agent pod in the **Running** state, the agent is running and collecting Prometheus metrics. By default the CloudWatch agent collects metrics for App Mesh, NGINX, Memcached, Java/JMX, and HAProxy every minute. For more information about those metrics, see [Prometheus metrics collected by the CloudWatch agent \(p. 440\)](#). For instructions on how to see your Prometheus metrics in CloudWatch, see [Viewing your Prometheus metrics \(p. 447\)](#)

You can also configure the CloudWatch agent to collect metrics from other Prometheus exporters. For more information, see [Scraping additional Prometheus sources and importing those metrics \(p. 418\)](#).

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in [`<scrape_config>`](#) in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon EKS clusters, the configurations are defined in `prometheus-eks.yaml` (for the EC2 launch type) or `prometheus-eks-fargate.yaml` (for the Fargate launch type) as two config maps:

- The `name: prometheus-config` section contains the settings for Prometheus scraping.
- The `name: prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent. You can use this section to configure how the Prometheus metrics are collected by CloudWatch. For example, you specify which metrics are to be imported into CloudWatch, and define their dimensions.

For Kubernetes clusters running on Amazon EC2 instances, the configurations are defined in the `prometheus-k8s.yaml` YAML file as two config maps:

- The `name: prometheus-config` section contains the settings for Prometheus scraping.
- The `name: prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in [`<scrape_config>`](#) in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:  
  scrape_interval: 1m  
  scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in `prometheus-eks.yaml`, the default scraping jobs are configured in the `job_name` lines in the `scrape_configs` section. In this file, the following default `kubernetes-pod-jmx` section scrapes JMX exporter metrics.

```
- job_name: 'kubernetes-pod-jmx'  
  sample_limit: 10000  
  metrics_path: /metrics  
  kubernetes_sd_configs:  
    - role: pod  
      relabel_configs:  
        - source_labels: [__address__]  
          action: keep  
          regex: '.*:9404$'  
        - action: labelmap  
          regex: __meta_kubernetes_pod_label_(.+)  
        - action: replace  
          source_labels:  
            - __meta_kubernetes_namespace  
          target_label: Namespace  
        - source_labels: [__meta_kubernetes_pod_name]  
          action: replace  
          target_label: pod_name  
        - action: replace  
          source_labels:  
            - __meta_kubernetes_pod_container_name  
          target_label: container_name  
        - action: replace  
          source_labels:  
            - __meta_kubernetes_pod_controller_name  
          target_label: pod_controller_name  
        - action: replace
```

```
source_labels:  
- __meta_kubernetes_pod_controller_kind  
target_label: pod_controller_kind  
- action: replace  
source_labels:  
- __meta_kubernetes_pod_phase  
target_label: pod_phase
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

Log events from Amazon EKS and Kubernetes clusters are stored in the `/aws/containerinsights/cluster_name/prometheus` log group in CloudWatch Logs. Log events from Amazon ECS clusters are stored in the `/aws/ecs/containerinsights/cluster_name/prometheus` log group.

Each scraping job is contained in a different log stream in this log group. For example, the Prometheus scraping job `kubernetes-pod-appmesh-envoy` is defined for App Mesh. All App Mesh Prometheus metrics from Amazon EKS and Kubernetes clusters are sent to the log stream named `/aws/containerinsights/cluster_name>prometheus/kubernetes-pod-appmesh-envoy/`.

To add a new scraping target, you add a new `job_name` section to the `scrape_configs` section of the YAML file, and restart the agent. For an example of this process, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics \(p. 422\)](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **`cluster_name`**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon EKS or Kubernetes cluster name.
- **`log_group_name`**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses `/aws/containerinsights/cluster_name/prometheus` for logs from Amazon EKS and Kubernetes clusters.
- **`prometheus_config_path`**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **`ecs_service_discovery`**— is the section to specify the configuration for Amazon ECS Prometheus service discovery. For more information, see [Detailed guide for autodiscovery on Amazon ECS clusters \(p. 384\)](#).

The `ecs_service_discovery` section can contain the following fields:

- `sd_frequency` is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

This field is optional. The default is 60 seconds (1 minute).

- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed..
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.

- `docker_label` is an optional section that you can use to specify the configuration for docker label-based service discovery. If you omit this section, docker label-based discovery is not used. This section can contain the following fields:
 - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this docker label, the CloudWatch agent will skip it.
 - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this docker label, the the agent assumes the default path `/metrics`.
 - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
 - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
 - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
 - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
 - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
 - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
 - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

If you have multiple labels specified in `source_labels`, we recommend that you do not use ^ or \$ characters in the regular expression for `label_matcher`.

- `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
- `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is ;. You can see this default used in the `label_matcher` line in the following example.
- `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
- `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [  
    {  
        "source_labels": [ "Service", "Namespace"],  
        "label_matcher": "(.*node-exporter.*|.*kube-dns.*);kube-system",  
        "dimensions": [  
            ["Service", "Namespace"]  
        ],  
        "metric_selectors": [  
            "^\^coredns_dns_request_type_count_total$"  
        ]  
    }]
```

```
    ] }
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{  
  "CloudWatchMetrics": [  
    {  
      "Metrics": [  
        {  
          "Name": "coredns_dns_request_type_count_total"  
        }  
      ],  
      "Dimensions": [  
        [  
          "Namespace",  
          "Service"  
        ]  
      ],  
      "Namespace": "ContainerInsights/Prometheus"  
    }  
  ],  
  "Namespace": "kube-system",  
  "Service": "kube-dns",  
  "coredns_dns_request_type_count_total": 2562,  
  "eks_amazonaws_com_component": "kube-dns",  
  "instance": "192.168.61.254:9153",  
  "job": "kubernetes-service-endpoints",  
  ...  
}
```

Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics

The Kubernetes API Server exposes Prometheus metrics on endpoints by default. The official example for the Kubernetes API Server scraping configuration is available on [Github](#).

The following tutorial shows how to do the following steps to begin importing Kubernetes API Server metrics into CloudWatch:

- Adding the Prometheus scraping configuration for Kubernetes API Server to the CloudWatch agent YAML file.
- Configuring the embedded metric format metrics definitions in the CloudWatch agent YAML file.
- (Optional) Creating a CloudWatch dashboard for the Kubernetes API Server metrics.

Note

The Kubernetes API Server exposes gauge, counter, histogram, and summary metrics. In this release of Prometheus metrics support, CloudWatch imports only the metrics with gauge, counter, and summary types.

To start collecting Kubernetes API Server Prometheus metrics in CloudWatch

1. Download the latest version of the `prometheus-eks.yaml`, `prometheus-eks-fargate.yaml`, or `prometheus-k8s.yaml` file by entering one of the following commands.

For an Amazon EKS cluster with the EC2 launch type, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

For a Kubernetes cluster running on an Amazon EC2 instance, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, find the `prometheus-config` section, and add the following section inside of that section. Then save the changes:

```
# Scrape config for API servers
- job_name: 'kubernetes-apiservers'
  kubernetes_sd_configs:
    - role: endpoints
      namespaces:
        names:
          - default
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
        - source_labels: [__meta_kubernetes_service_name,
                      __meta_kubernetes_endpoint_port_name]
          action: keep
          regex: kubernetes;https
        - action: replace
          source_labels:
            - __meta_kubernetes_namespace
          target_label: Namespace
        - action: replace
          source_labels:
            - __meta_kubernetes_service_name
          target_label: Service
```

3. While you still have the YAML file open in the text editor, find the `cwagentconfig.json` section. Add the following subsection and save the changes. This section puts the API server metrics onto the CloudWatch agent allow list. Three types of API Server metrics are added to the allow list:

- etcd object counts
- API Server registration controller metrics
- API Server request metrics

```
{"source_labels": ["job", "resource"],  
 "label_matcher": "^kubernetes-apiservers;(services|daemonsets.apps|deployments.apps|configmaps|endpoints|secrets|serviceaccounts|replicasets.apps)",  
 "dimensions": [["ClusterName","Service","resource"]],  
 "metric_selectors": [  
 "^\$etcd_object_counts$"  
 ]  
,  
 {"source_labels": ["job", "name"],  
 "label_matcher": "^kubernetes-apiservers;APIServiceRegistrationController$",  
 "dimensions": [["ClusterName","Service","name"]],  
 "metric_selectors": [  
 "^\$workqueue_depth$",  
 "^\$workqueue_adds_total$",  
 "^\$workqueue_retries_total$"  
 ]  
,  
 {"source_labels": ["job", "code"],  
 "label_matcher": "^kubernetes-apiservers;2[0-9]{2}$",  
 "dimensions": [["ClusterName","Service","code"]],  
 "metric_selectors": [  
 "^\$apiserver_request_total$"  
 ]  
,  
 {"source_labels": ["job"],  
 "label_matcher": "^kubernetes-apiservers",  
 "dimensions": [["ClusterName","Service"]],  
 "metric_selectors": [  
 "^\$apiserver_request_total$"  
 ]  
,
```

4. If you already have the CloudWatch agent with Prometheus support deployed in the cluster, you must delete it by entering the following command:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

5. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter:

```
kubectl apply -f prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-eks-fargate.yaml \  
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \  
| kubectl apply -f -
```

For a Kubernetes cluster, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-k8s.yaml \  
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \  
| kubectl apply -f -
```

Once you have done this, you should see a new log stream named **kubernetes-apiservers** in the **/aws/containerinsights/*cluster_name*/prometheus** log group. This log stream should include log events with an embedded metric format definition like the following:

```
{  
    "CloudWatchMetrics": [  
        {  
            "Metrics": [  
                {  
                    "Name": "apiserver_request_total"  
                }  
            ],  
            "Dimensions": [  
                [  
                    "ClusterName",  
                    "Service"  
                ]  
            ],  
            "Namespace": "ContainerInsights/Prometheus"  
        }  
    ],  
    "ClusterName": "my-cluster-name",  
    "Namespace": "default",  
    "Service": "kubernetes",  
    "Timestamp": "1592267020339",  
    "Version": "0",  
    "apiserver_request_count": 0,  
    "apiserver_request_total": 0,  
    "code": "0",  
    "component": "apiserver",  
    "contentType": "application/json",  
    "instance": "192.0.2.0:443",  
    "job": "kubernetes-apiservers",  
    "prom_metric_type": "counter",  
    "resource": "pods",  
    "scope": "namespace",  
    "verb": "WATCH",  
    "version": "v1"  
}
```

You can view your metrics in the CloudWatch console in the **ContainerInsights/Prometheus** namespace. You can also optionally create a CloudWatch dashboard for your Prometheus Kubernetes API Server metrics.

(Optional) Creating a dashboard for Kubernetes API Server metrics

To see Kubernetes API Server metrics in your dashboard, you must have first completed the steps in the previous sections to start collecting these metrics in CloudWatch.

To create a dashboard for Kubernetes API Server metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Make sure you have the correct AWS Region selected.
3. In the navigation pane, choose **Dashboards**.
4. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
5. In **Add to this dashboard**, choose **Cancel**.
6. Choose **Actions, View/edit source**.
7. Download the following JSON file: [Kubernetes API Dashboard source](#).
8. Open the JSON file that you downloaded with a text editor, and make the following changes:

- Replace all the {{YOUR_CLUSTER_NAME}} strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.
 - Replace all the {{YOUR_AWS_REGION}} strings with the name of the Region where the metrics are collected. For example us-west-2. Be sure not to add whitespaces before or after the text.
9. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
 10. Choose **Update, Save dashboard**.

(Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see [Prometheus metrics collected by the CloudWatch agent \(p. 440\)](#).

Before you can install any of these workloads, you must install Helm 3.x by entering the following commands:

```
brew install helm
```

For more information, see [Helm](#).

Topics

- [Set up AWS App Mesh sample workload for Amazon EKS and Kubernetes \(p. 426\)](#)
- [Set up NGINX with sample traffic on Amazon EKS and Kubernetes \(p. 429\)](#)
- [Set up memcached with a metric exporter on Amazon EKS and Kubernetes \(p. 430\)](#)
- [Set up Java/JMX sample workload on Amazon EKS and Kubernetes \(p. 431\)](#)
- [Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes \(p. 435\)](#)
- [Tutorial for adding a new Prometheus scrape target: Redis on Amazon EKS and Kubernetes clusters \(p. 436\)](#)

Set up AWS App Mesh sample workload for Amazon EKS and Kubernetes

Prometheus support in CloudWatch Container Insights supports AWS App Mesh. The following sections explain how to set up App Mesh.

CloudWatch Container Insights can also collect App Mesh Envoy Access Logs. For more information, see [\(Optional\) Enable App Mesh Envoy access logs \(p. 339\)](#).

Topics

- [Set up AWS App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster \(p. 426\)](#)
- [Set up AWS App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type \(p. 428\)](#)

Set up AWS App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the EC2 launch type, or a Kubernetes cluster.

Configure IAM permissions

You must add the **AWSAppMeshFullAccess** policy to the IAM role for your Amazon EKS or Kubernetes node group. On Amazon EKS, this node group name looks similar to `eksctl-integ-test-eks-prometheus-NodeInstanceRole-ABCDEFHIJKL`. On Kubernetes, it might look similar to `nodes.integ-test-kops-prometheus.k8s.local`.

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in [App Mesh Controller](#).

Install a sample application

[aws-app-mesh-examples](#) contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that shows how http routes can use headers for matching incoming requests.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: <https://github.com/aws/aws-app-mesh-examples/tree/master/walkthroughs/howto-k8s-http-headers>.
2. Launch a curler pod to generate traffic:

```
kubectl -n default run -it curler --image=tutum/curl /bin/bash
```
3. Curl different endpoints by changing HTTP headers. Run the curl command multiple times, as shown:

```
curl -H "color_header: blue" front.howto-k8s-http-headers.svc.cluster.local:8080/; echo;
curl -H "color_header: red" front.howto-k8s-http-headers.svc.cluster.local:8080/; echo;
curl -H "color_header: yellow" front.howto-k8s-http-headers.svc.cluster.local:8080/; echo;
```
4. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
5. In the AWS Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-http-headers/
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up AWS App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the Fargate launch type.

Configure IAM permissions

To set up IAM permissions, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create iamserviceaccount --cluster MyCluster \
--namespace howto-k8s-fargate \
--name appmesh-pod \
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
--override-existing-serviceaccounts \
--approve
```

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in [App Mesh Controller](#). Be sure to follow the instructions for Amazon EKS with the Fargate launch type.

Install a sample application

[aws-app-mesh-examples](#) contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that works for Amazon EKS clusters with the Fargate launch type.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: <https://github.com/aws/aws-app-mesh-examples/tree/master/walkthroughs/howto-k8s-fargate>.

Those instructions assume that you are creating a new cluster with the correct Fargate profile. If you want to use an Amazon EKS cluster that you've already set up, you can use the following commands to set up that cluster for this demonstration. Replace *MyCluster* with the name of your cluster.

```
eksctl create iamserviceaccount --cluster MyCluster \
--namespace howto-k8s-fargate \
--name appmesh-pod \
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
--override-existing-serviceaccounts \
--approve
```

```
eksctl create fargateprofile --cluster MyCluster \
--namespace howto-k8s-fargate --name howto-k8s-fargate
```

2. Port forward the front application deployment:

```
kubectl -n howto-k8s-fargate port-forward deployment/front 8080:8080
```

3. Curl the front app:

```
while true; do curl -s http://localhost:8080/color; sleep 0.1; echo ; done
```

4. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
5. In the AWS Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-fargate/
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up NGINX with sample traffic on Amazon EKS and Kubernetes

NGINX is a web server that can also be used as a load balancer and reverse proxy. For more information about how Kubernetes uses NGINX for ingress , see [kubernetes/ingress-nginx](#).

To install Ingress-NGINX with a sample traffic service to test Container Insights Prometheus support

1. Enter the following command to add the Helm ingress-nginx repo:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

2. Enter the following commands:

```
kubectl create namespace nginx-ingress-sample

helm install my-inginx ingress-nginx/ingress-nginx \
--namespace nginx-ingress-sample \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="10254" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

3. Check whether the services started correctly by entering the following command:

```
kubectl get service -n nginx-ingress-sample
```

The output of this command should display several columns, including an EXTERNAL-IP column.

4. Set an EXTERNAL-IP variable to the value of the EXTERNAL-IP column in the row of the NGINX ingress controller.

```
EXTERNAL_IP=your-inginx-controller-external-ip
```

5. Start some sample NGINX traffic by entering the following command.

```
SAMPLE_TRAFFIC_NAMESPACE=nginx-sample-traffic
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_traffic/nginx-traffic/nginx-traffic-sample.yaml |
sed "s/{{external_ip}}/$EXTERNAL_IP/g" |
sed "s/{{namespace}}/$SAMPLE_TRAFFIC_NAMESPACE/g" |
kubectl apply -f -
```

6. Enter the following command to confirm that all three pods are in the Running status.

```
kubectl get pod -n $SAMPLE_TRAFFIC_NAMESPACE
```

If they are running, you should soon see metrics in the **ContainerInsights/Prometheus** namespace.

To uninstall NGINX and the sample traffic application

1. Delete the sample traffic service by entering the following command:

```
kubectl delete namespace $SAMPLE_TRAFFIC_NAMESPACE
```

2. Delete the NGINX egress by the Helm release name.

```
helm uninstall my-nginx --namespace nginx-ingress-sample
kubectl delete namespace nginx-ingress-sample
```

Set up memcached with a metric exporter on Amazon EKS and Kubernetes

memcached is an open-source memory object caching system. For more information, see [What is Memcached?](#).

If you are running memcached on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace **MyCluster** with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace memcached-sample --name memcached-sample
```

To install memcached with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the repo:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace memcached-sample
```

3. Enter the following command to install Memcached

```
helm install my-memcached bitnami/memcached --namespace memcached-sample \
--set metrics.enabled=true \
--set-string serviceAnnotations.prometheus\\.io/port="9150" \
--set-string serviceAnnotations.prometheus\\.io/scrape="true"
```

4. Enter the following command to confirm the annotation of the running service:

```
kubectl describe service my-memcached-metrics -n memcached-sample
```

You should see the following two annotations:

```
Annotations:  prometheus.io/port: 9150
              prometheus.io/scrape: true
```

To uninstall memcached

- Enter the following commands:

```
helm uninstall my-memcached --namespace memcached-sample
kubectl delete namespace memcached-sample
```

Set up Java/JMX sample workload on Amazon EKS and Kubernetes

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see [prometheus/jmx_exporter](#).

Container Insights can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter.

Default Prometheus scrape configuration

By default, the CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics from `http://CLUSTER_IP:9404/metrics` on each pod in an Amazon EKS or Kubernetes cluster. This is done by `role: pod` discovery of Prometheus `kubernetes_sd_config`. 9404 is the default port allocated for JMX Exporter by Prometheus. For more information about `role: pod` discovery, see [pod](#). You can configure the JMX Exporter to expose the metrics on a different port or `metrics_path`. If you do change the port or path, update the default `jmx scrape_config` in the CloudWatch agent config map. Run the following command to get the current CloudWatch agent Prometheus configuration:

```
kubectl describe cm prometheus-config -n amazon-cloudwatch
```

The fields to change are the `/metrics` and `regex: '.*:9404$'` fields, as highlighted in the following example.

```
job_name: 'kubernetes-jmx-pod'
sample_limit: 10000
metrics_path: /metrics
kubernetes_sd_configs:
- role: pod
relabel_configs:
- source_labels: [__address__]
  action: keep
  regex: '.*:9404$'
- action: replace
  regex: (.+)
  source_labels:
```

Other Prometheus scrape configuration

If you expose your application running on a set of pods with Java/JMX Prometheus exporters by a Kubernetes Service, you can also switch to use `role: service discovery` or `role: endpoint`

discovery of Prometheus `kubernetes_sd_config`. For more information about these discovery methods, see [service](#), [endpoints](#), and [`<kubernetes_sd_config>`](#).

More meta labels are provided by these two service discovery modes which could be useful for you to build the CloudWatch metrics dimensions. For example, you can relabel `_meta_kubernetes_service_name` to `Service` and include it into your metrics' dimension. For more information about customizing your CloudWatch metrics and their dimensions, see [CloudWatch agent configuration for Prometheus \(p. 381\)](#).

Docker image with JMX Exporter

Next, build a Docker image. The following sections provide two example Dockerfiles.

When you have built the image, load it into Amazon EKS or Kubernetes, and then run the following command to verify that Prometheus metrics are exposed by `JMX_EXPORTER` on port 9404. Replace `$JAR_SAMPLE_TRAFFIC_POD` with the running pod name and replace `$JAR_SAMPLE_TRAFFIC_NAMESPACE` with your application namespace.

If you are running JMX Exporter on a cluster with the Fargate launch type, you also need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace `MyCluster` with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $JAR_SAMPLE_TRAFFIC_NAMESPACE \
--name $JAR_SAMPLE_TRAFFIC_NAMESPACE
```

```
kubectl exec $JAR_SAMPLE_TRAFFIC_POD -n $JARCAT_SAMPLE_TRAFFIC_NAMESPACE -- curl http://
localhost:9404
```

Example: Apache Tomcat Docker image with Prometheus metrics

Apache Tomcat server exposes JMX mBeans by default. You can integrate JMX Exporter with Tomcat to expose JMX mBeans as Prometheus metrics. The following example Dockerfile shows the steps to build a testing image:

```
# From Tomcat 9.0 JDK8 OpenJDK
FROM tomcat:9.0-jdk8-openjdk

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter
COPY ./setenv.sh /usr/local/tomcat/bin
COPY your web application.war /usr/local/tomcat/webapps/

RUN chmod o+x /usr/local/tomcat/bin/setenv.sh

ENTRYPOINT ["catalina.sh", "run"]
```

The following list explains the four `COPY` lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

Here is a sample configuration file for Java and Tomcat:

```
lowercaseOutputName: true
```

```

lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name="\\"(\w+-\w+)-(\d+)\\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
    help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=/([-a-zA-Z0-9+\&#/%=~_!:\.,;]*[-a-zA-Z0-9+\&#/%=~_!|]), name=([-a-zA-Z0-9+/$_-|!.]*)', J2EEApplication=none, J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
    help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="\\"(\w+-\w+)-(\d+)\\"><>(currentThreadCount|currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
    help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+\&#/%=~_!:\.,;]*[-a-zA-Z0-9+\&#/%=~_!|]), context=([-a-zA-Z0-9+/$_-|!.]*)><>(processingTime|sessionCounter|rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
    help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"

```

- `setenv.sh` is a Tomcat startup script to start the JMX exporter along with Tomcat and expose Prometheus metrics on port 9404 of the localhost. It also provides the JMX Exporter with the `config.yaml` file path.

```

$ cat setenv.sh
export JAVA_OPTS="-javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml $JAVA_OPTS"

```

- your web application.war is your web application war file to be loaded by Tomcat.

Build a Docker image with this configuration and upload it to an image repository.

Example: Java Jar Application Docker image with Prometheus metrics

The following example Dockerfile shows the steps to build a testing image:

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./SampleJavaApplication-1.0-SNAPSHOT.jar /opt/jmx_exporter
COPY ./start_exporter_example.sh /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter

RUN chmod -R o+x /opt/jmx_exporter
RUN apk add curl

ENTRYPOINT exec /opt/jmx_exporter/start_exporter_example.sh
```

The following list explains the four COPY lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- config.yaml is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

Here is a sample configuration file for Java and Tomcat:

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=(\w+-\w+)-(\d+)\">'><>(\w+)
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=/([-a-zA-Z0-9+\&@#/%?=~_|!:,;]*[-a-zA-Z0-9+\&@#/%=~_|]), name=([-a-zA-Z0-9+/$_-|!.]*)', J2EEApplication=none, J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
```

```
port: "$2"
protocol: "$1"
help: Catalina threadpool $3
type: GAUGE

- pattern: 'Catalina<type=Manager, host=(-a-zA-Z0-9+&@#/%?=~-|!:.,;]*[-a-zA-Z0-9+&@#/%=-_| ]), context=(-a-zA-Z0-9+/$%-|-|.)*><>(processingTime|sessionCounter|rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `start_exporter_example.sh` is the script to start the JAR application with the Prometheus metrics exported. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat start_exporter_example.sh
java -javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml -cp /opt/jmx_exporter/SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

- `SampleJavaApplication-1.0-SNAPSHOT.jar` is the sample Java application jar file. Replace it with the Java application that you want to monitor.

Build a Docker image with this configuration and upload it to an image repository.

Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes

HAProxy is an open-source proxy application. For more information, see [HAProxy](#).

If you are running HAProxy on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace `MyCluster` with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace haproxy-ingress-sample --name haproxy-ingress-sample
```

To install HAProxy with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the Helm incubator repo:

```
helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace haproxy-ingress-sample
```

3. Enter the following commands to install HAProxy:

```
helm install haproxy haproxy-ingress/haproxy-ingress \
--namespace haproxy-ingress-sample \
--set defaultBackend.enabled=true \
--set controller.stats.enabled=true \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="9101" \
```

```
--set-string controller.metrics.service.annotations."prometheus\\.io/scrape"="true"
```

4. Enter the following command to confirm the annotation of the service:

```
kubectl describe service haproxy-haproxy-ingress-metrics -n haproxy-ingress-sample
```

You should see the following annotations.

```
Annotations:  prometheus.io/port: 9101  
              prometheus.io/scrape: true
```

To uninstall HAProxy

- Enter the following commands:

```
helm uninstall haproxy --namespace haproxy-ingress-sample  
kubectl delete namespace haproxy-ingress-sample
```

Tutorial for adding a new Prometheus scrape target: Redis on Amazon EKS and Kubernetes clusters

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis application on Amazon EKS and Kubernetes. Redis (<https://redis.io/>) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see [redis](#).

`redis_exporter` (MIT License licensed) is used to expose the Redis prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see [redis_exporter](#).

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [redis](#)
- [redis_exporter](#)

To install a sample Redis workload which exposes Prometheus metrics

1. Set the namespace for the sample Redis workload.

```
REDIS_NAMESPACE=redis-sample
```

2. If you are running Redis on a cluster with the Fargate launch type, you need to set up a Fargate profile. To set up the profile, enter the following command. Replace `MyCluster` with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \  
--namespace $REDIS_NAMESPACE --name $REDIS_NAMESPACE
```

3. Enter the following command to install the sample Redis workload.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml \  
| sed "s/{{namespace}}/$REDIS_NAMESPACE/g" \  
| kubectl apply -f -
```

4. The installation includes a service named `my-redis-metrics` which exposes the Redis Prometheus metric on port 9121. Enter the following command to get the details of the service:

```
kubectl describe service/my-redis-metrics -n $REDIS_NAMESPACE
```

In the Annotations section of the results, you'll see two annotations which match the Prometheus scrape configuration of the CloudWatch agent, so that it can auto-discover the workloads:

```
prometheus.io/port: 9121
prometheus.io/scrape: true
```

The related Prometheus scrape configuration can be found in the `- job_name: kubernetes-service-endpoints` section of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml`.

To start collecting Redis Prometheus metrics in CloudWatch

1. Download the latest version of the `kubernetes-eks.yaml` or `kubernetes-k8s.yaml` file by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

For a Kubernetes cluster running on an Amazon EC2 instance, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, and find the `cwagentconfig.json` section. Add the following subsection and save the changes. Be sure that the indentation follows the existing pattern.

```
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace", "ClusterName"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ],
  {
    "source_labels": ["pod_name"],
    "label_matcher": "^redis-instance$",
    "dimensions": [["Namespace", "ClusterName", "cmd"]],
    "metric_selectors": [
      "^redis_commands_total$"
    ]
  }
}
```

```

        ],
    },
{
    "source_labels": ["pod_name"],
    "label_matcher": "^redis-instance$",
    "dimensions": [["Namespace", "ClusterName", "db"]],
    "metric_selectors": [
        "^redis_db_keys$"
    ]
},

```

The section you added puts the Redis metrics onto the CloudWatch agent allow list. For a list of these metrics, see the following section.

3. If you already have the CloudWatch agent with Prometheus support deployed in this cluster, you must delete it by entering the following command.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

4. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. Replace *MyCluster* and *region* to match your settings.

For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
kubectl apply -f prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

For a Kubernetes cluster, enter this command.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Viewing your Redis Prometheus metrics

This tutorial sends the following metrics to the **ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions
redis_net_input_byte	ClusterName, Namespace
redis_net_output_byte	ClusterName, Namespace
redis_expired_keys_t6	ClusterName, Namespace
redis_evicted_keys_t6	ClusterName, Namespace
redis_keyspace_hits	ClusterName, Namespace
redis_keyspace_misses	ClusterName, Namespace
redis_memory_used_by	ClusterName, Namespace

Metric name	Dimensions	
redis_connected_clients	ClusterName, Namespace	
redis_commands_total	ClusterName, Namespace, cmd	
redis_db_keys	ClusterName, Namespace, db	

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.

The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis Prometheus metrics.

To create a dashboard for Redis Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-east-1
CLUSTER_NAME=your_k8s_cluster_name_here
NAMESPACE=your_redis_service_namespace_here
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/master/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Prometheus metric type conversion by the CloudWatch Agent

The Prometheus client libraries offer four core metric types:

- Counter
- Gauge
- Summary
- Histogram

The CloudWatch agent supports the counter, gauge, and summary metric types. Support for histogram metrics is planned for an upcoming release.

The Prometheus metrics with the unsupported histogram metric type are dropped by the CloudWatch agent. For more information, see [Logging dropped Prometheus metrics \(p. 452\)](#).

Gauge metrics

A Prometheus gauge metric is a metric that represents a single numerical value that can arbitrarily go up and down. The CloudWatch agent scrapes gauge metrics and send these values out directly.

Counter metrics

A Prometheus counter metric is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero. The CloudWatch agent calculates a delta from the previous scrape and sends the delta value as the metric value in the log event. So the CloudWatch agent will start to produce one log event from the second scrape and continue with subsequent scrapes, if any.

Summary metrics

A Prometheus summary metric is a complex metric type which is represented by multiple data points. It provides a total count of observations and a sum of all observed values. It calculates configurable quantiles over a sliding time window.

The sum and count of a summary metric are cumulative, but the quantiles are not. The following example shows the variance of quantiles.

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 7.123e-06
go_gc_duration_seconds{quantile="0.25"} 9.204e-06
go_gc_duration_seconds{quantile="0.5"} 1.1065e-05
go_gc_duration_seconds{quantile="0.75"} 2.8731e-05
go_gc_duration_seconds{quantile="1"} 0.003841496
go_gc_duration_seconds_sum 0.37630427
go_gc_duration_seconds_count 9774
```

The CloudWatch agent handles the sum and count of a summary metric in the same way as it handles counter metrics, as described in the previous section. The CloudWatch agent preserves the quantile values as they are originally reported.

Prometheus metrics collected by the CloudWatch agent

The CloudWatch agent with Prometheus support automatically collects metrics from several services and workloads. The metrics that are collected by default are listed in the following sections. You can also configure the agent to collect more metrics from these services, and to collect Prometheus metrics from other applications and services. For more information about collecting additional metrics, see [CloudWatch agent configuration for Prometheus \(p. 381\)](#).

Prometheus metrics collected from Amazon EKS and Kubernetes clusters are in the **ContainerInsights/Prometheus** namespace. Prometheus metrics collected from Amazon ECS clusters are in the **ECS/ContainerInsights/Prometheus** namespace.

Topics

- [Prometheus metrics for App Mesh \(p. 440\)](#)
- [Prometheus metrics for NGINX \(p. 443\)](#)
- [Prometheus metrics for Memcached \(p. 443\)](#)
- [Prometheus metrics for Java/JMX \(p. 444\)](#)
- [Prometheus metrics for HAProxy \(p. 446\)](#)

Prometheus metrics for App Mesh

The following metrics are automatically collected from App Mesh .

CloudWatch Container Insights can also collect App Mesh Envoy Access Logs. For more information, see [\(Optional\) Enable App Mesh Envoy access logs \(p. 339\)](#).

Prometheus metrics for App Mesh on Amazon EKS and Kubernetes clusters

Metric name	Dimensions
envoy_http_downstream_rq	ClusterName, Namespace
envoy_http_downstream_rq	ClusterName, Namespace
	ClusterName, Namespace, envoy_http_conn_manager_prefix, envoy_response_code_class
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_member	ClusterName, Namespace
envoy_cluster_member	ClusterName, Namespace
envoy_server_memory_usage	ClusterName, Namespace
envoy_server_memory_usage	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace_eject
envoy_cluster_upstream_rq	ClusterName, Namespace_low
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace_with_active_rq
envoy_cluster_upstream_rq	ClusterName, Namespace_active_rq
envoy_cluster_upstream_rq	ClusterName, Namespace_code
envoy_cluster_upstream_rq	ClusterName, Namespace_total
envoy_cluster_upstream_rq	ClusterName, Namespace_total
envoy_cluster_upstream_rq	ClusterName, Namespace_parked_up_total
envoy_cluster_upstream_rq	ClusterName, Namespace_parked_total
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_cluster_upstream_rq	ClusterName, Namespace
envoy_server_live	ClusterName, Namespace
envoy_server_uptime	ClusterName, Namespace

Prometheus metrics for App Mesh on Amazon ECS clusters

Note

`TaskDefinitionFamily` is the Kubernetes namespace of the mesh.

The value of `envoy.http_conn_manager.prefix` can be `ingress`, `egress`, or `admin`.

The value of `envoy_response_code_class` can be 1xxx, 2xxx, or 5xxx. The value of `envoy_response_code_class` can be 1 (stands for 1xx), 2 stands for 2xx), 3 stands for 3xx), 4 stands for 4xx), or 5 stands for 5xx).

Prometheus metrics for NGINX

The following metrics are automatically collected from NGINX on Amazon EKS and Kubernetes clusters.

Prometheus metrics for Memcached

The following metrics are automatically collected from Memcached on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions
memcached_current_items	ClusterName, Namespace, Service
memcached_current_connections	ClusterName, Namespace, Service
memcached_limit_bytes	ClusterName, Namespace, Service
memcached_current_bytes	ClusterName, Namespace, Service
memcached_written_bytes	ClusterName, Namespace, Service
memcached_read_bytes	ClusterName, Namespace, Service
memcached_items_expired	ClusterName, Namespace, Service
memcached_items_recycled	ClusterName, Namespace, Service
memcached_commands_total	ClusterName, Namespace, Service, command
	ClusterName, Namespace, Service, status, command

Prometheus metrics for Java/JMX

Metrics collected on Amazon EKS and Kubernetes clusters

On Amazon EKS and Kubernetes clusters, Container Insights can collect the following predefined Prometheus metrics from the Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter. For more information, see [prometheus/jmx_exporter](#) on Github.

Java/JMX on Amazon EKS and Kubernetes clusters

Metric name	Dimensions
jvm_classes_loaded	ClusterName, Namespace
jvm_threads_current	ClusterName, Namespace
jvm_threads_daemon	ClusterName, Namespace
java_lang_operatingsystemsize	ClusterName, Namespace
java_lang_operatingsystemloadavg	ClusterName, Namespace
java_lang_operatingsystemcpuused	ClusterName, Namespace
java_lang_operatingsystemcpusize	ClusterName, Namespace
java_lang_operatingsystemmemorysize	ClusterName, Namespace
java_lang_operatingsystemmemoriesize	ClusterName, Namespace
java_lang_operatingsystemprocesscount	ClusterName, Namespace
java_lang_operatingsystemprocessors	ClusterName, Namespace
jvm_memory_bytes_use	ClusterName, Namespace, area
jvm_memory_pool_byte	ClusterName, Namespace, pool

Note

The values of the area dimension can be heap or nonheap.

The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon EKS and Kubernetes clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload.

Metric name	Dimensions
catalina_manager_activeclusters	ClusterName, Namespace
catalina_manager_rejectedclusters	ClusterName, Namespace
catalina_globalrequestcount	ClusterName, Namespace, received
catalina_globalrequestcountsent	ClusterName, Namespace, sent
catalina_globalrequestcountcount	ClusterName, Namespace, count

Metric name	Dimensions
catalina_globalrequestcount	ClusterName, Namespace
catalina_globalrequestingtime	ClusterName, Namespace

Java/JMX on Amazon ECS clusters

Metric name	Dimensions
jvm_classes_loaded	ClusterName, TaskDefinitionFamily
jvm_threads_current	ClusterName, TaskDefinitionFamily
jvm_threads_daemon	ClusterName, TaskDefinitionFamily
java_lang_operatingsystemcpu	ClusterName, TaskDefinitionFamily
java_lang_operatingsystemmemory	ClusterName, TaskDefinitionFamily
java_lang_operatingsystemprocesses	ClusterName, TaskDefinitionFamily
java_lang_operatingsystemthreads	ClusterName, TaskDefinitionFamily
java_lang_operatingsystemvirtualmemsize	ClusterName, TaskDefinitionFamily
jvm_memory_bytes_usage	ClusterName, TaskDefinitionFamily, area
jvm_memory_pool_bytes	ClusterName, TaskDefinitionFamily, pool

Note

The values of the area dimension can be heap or nonheap.

The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon ECS clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload on Amazon ECS clusters.

Metric name	Dimensions
catalina_manager_activeconnections	ClusterName, TaskDefinitionFamily
catalina_manager_rejectedconnections	ClusterName, TaskDefinitionFamily
catalina_globalrequestcount	ClusterName, TaskDefinitionFamily
catalina_globalrequestingtime	ClusterName, TaskDefinitionFamily
catalina_globalrequesttime	ClusterName, TaskDefinitionFamily

Metric name	Dimensions
catalina_globalrequest	ClusterName, TaskDefinitionFamily
catalina_globalrequest	ClusterName, TaskDefinitionFamily

Prometheus metrics for HAProxy

The following metrics are automatically collected from HAProxy on Amazon EKS and Kubernetes clusters.

The metrics collected depend on which version of HAProxy Ingress that you are using. For more information about HAProxy Ingress and its versions, see [haproxy-ingress](#).

Metric name	Dimensions	Availability
haproxy_backend_byte	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_backend_byte	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_backend_connection	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_backend_connection	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_backend_current	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_backend_http	ClusterName, Namespace, Service, code, backend	All versions of HAProxy Ingress
haproxy_backend_stat	ClusterName, Namespace, Service	Only in versions 0.10 or later of HAProxy Ingress
haproxy_backend_up	ClusterName, Namespace, Service	Only in versions of HAProxy Ingress earlier than 0.10
haproxy_frontend_byte	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_frontend_byte	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_frontend_connection	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_frontend_current	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_frontend_http	ClusterName, Namespace, Service	All versions of HAProxy Ingress
haproxy_frontend_http	ClusterName, Namespace, Service, code, frontend	All versions of HAProxy Ingress

Metric name	Dimensions	Availability
haproxy_frontend_req	Cluster Name, Namespace, Service	All versions of HAProxy Ingress
haproxy_frontend_req	Cluster Name, Namespace, Service	All versions of HAProxy Ingress

Note

The values of the code dimension can be 1xx, 2xx, 3xx, 4xx, 5xx, or other.

The values of the backend dimension can be:

- http-default-backend, http-shared-backend, or httpsback-shared-backend for HAProxy Ingress version 0.0.27 or earlier.
- _default_backend for HAProxy Ingress versions later than 0.0.27.

The values of the frontend dimension can be:

- httpfront-default-backend, httpfront-shared-frontend, or httpfronts for HAProxy Ingress version 0.0.27 or earlier.
- _front_http or _front_https for HAProxy Ingress versions later than 0.0.27.

Viewing your Prometheus metrics

You can monitor and alarm on all your Prometheus metrics including the curated pre-aggregated metrics from App Mesh, NGINX, Java/JMX, Memcached, and HAProxy, and any other manually configured Prometheus exporter you may have added. For more information about collecting metrics from other Prometheus exporters, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics \(p. 422\)](#).

In the CloudWatch console, Container Insights provides the following pre-built reports:

- For Amazon EKS and Kubernetes clusters, there are pre-built reports for App Mesh, NGINX, HAProxy, Memcached, and Java/JMX.
- For Amazon ECS clusters, there are pre-built reports for App Mesh and Java/JMX.

Container Insights also provides custom dashboards for each of the workloads that Container Insights collects curated metrics from. You can download these dashboards from GitHub

To see all your Prometheus metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the list of namespaces, choose **ContainerInsights/Prometheus** or **ECS/ContainerInsights/Prometheus**.
4. Choose one of the sets of dimensions in the following list. Then select the checkbox next to the metrics that you want to see.

To see pre-built reports on your Prometheus metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Performance Monitoring**.

3. In the drop-down box near the top of the page, choose any of the Prometheus options.

In the other drop-down box, choose a cluster to view

We have also provided custom dashboards for NGINX, App Mesh, Memcached, HAProxy, and Java/JMX.

To use a custom dashboard that Amazon has provided

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
4. In **Add to this dashboard**, choose **Cancel**.
5. Choose **Actions, View/edit source**.
6. Download one of the following JSON files:
 - [NGINX custom dashboard source on Github](#).
 - [App Mesh custom dashboard source on Github](#).
 - [Memcached custom dashboard source on Github](#)
 - [HAProxy-Ingress custom dashboard source on Github](#)
 - [Java/JMX custom dashboard source on Github](#).
7. Open the JSON file that you downloaded with a text editor, and make the following changes:
 - Replace all the {{YOUR_CLUSTER_NAME}} strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.
 - Replace all the {{YOUR_REGION}} strings with the AWS Region where your cluster is running. For example, **us-west-1** Make sure not to add whitespaces before or after the text.
 - Replace all the {{YOUR_NAMESPACE}} strings with the exact namespace of your workload.
 - Replace all the {{YOUR_SERVICE_NAME}} strings with the exact service name of your workload. For example, **haproxy-haproxy-ingress-controller-metrics**
8. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
9. Choose **Update, Save dashboard**.

Prometheus metrics troubleshooting

This section provides help for troubleshooting your Prometheus metrics setup.

Topics

- [Prometheus metrics troubleshooting on Amazon ECS \(p. 448\)](#)
- [Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters \(p. 451\)](#)

Prometheus metrics troubleshooting on Amazon ECS

This section provides help for troubleshooting your Prometheus metrics setup on Amazon ECS clusters.

I don't see Prometheus metrics sent to CloudWatch Logs

The Prometheus metrics should be ingested as log events in the log group **/aws/ecs/containerinsights/cluster-name/Prometheus**. If the log group is not created or the Prometheus metrics are not sent to the log group, you will need to first check whether the Prometheus targets have been successfully discovered

by the CloudWatch agent. And next check the security group and permission settings of the CloudWatch agent. The following steps guide you to do the debugging.

Step 1: Enable the CloudWatch agent debugging mode

First, change the CloudWatch agent to debug mode by adding the following bold lines to your AWS CloudFormation template file, `cwagent-ecs-prometheus-metric-for-bridge-host.yaml` or `cwagent-ecs-prometheus-metric-for-awsvpc.yaml`. Then save the file.

```
cwagentconfig.json: |
  {
    "agent": {
      "debug": true
    },
    "logs": {
      "metrics_collected": {
```

Create a new AWS CloudFormation changeset against the existing stack. Set other parameters in the changeset to the same values as in your existing AWS CloudFormation stack. The following example is for a CloudWatch agent installed in an Amazon ECS cluster using the EC2 launch type and the bridge network mode.

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
NEW_CHANGESET_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
--parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME \
--capabilities CAPABILITY_NAMED_IAM \
--region $AWS_REGION \
--change-set-name $NEW_CHANGESET_NAME
```

Go to the AWS CloudFormation console to review the new changeset, `$NEW_CHANGESET_NAME`. There should be one change applied to the **CWAgentConfigSSMParameter** resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster ${ECS_CLUSTER_NAME} \
--desired-count 0 \
--service your_service_name_here \
--region $AWS_REGION
```

Wait about 10 seconds and then enter the following command.

```
aws ecs update-service --cluster ${ECS_CLUSTER_NAME} \
--desired-count 1 \
--service your_service_name_here \
--region $AWS_REGION
```

Step 2: Check the ECS service discovery logs

The ECS task definition of the CloudWatch agent enables the logs by default in the section below. The logs are sent to CloudWatch Logs in the log group `/ecs/ecs-cwagent-prometheus`.

```
LogConfiguration:  
  LogDriver: awslogs  
  Options:  
    awslogs-create-group: 'True'  
    awslogs-group: "/ecs/ecs-cwagent-prometheus"  
    awslogs-region: !Ref AWS::Region  
    awslogs-stream-prefix: !Sub 'ecs-${ECSLaunchType}-awsvpc'
```

Filter the logs by the string `ECS_SD_Stats` to get the metrics related to the ECS service discovery, as shown in the following example.

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeContainerInstances: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeInstancesRequest: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTaskDefinition: 2  
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTasks: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_ListTasks: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: Exporter_DiscoveredTargetCount: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_EC2MetaData: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_TaskDefinition: 2  
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_ContainerInstance: 1  
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_TaskDefinition: 2  
2020-09-1T01:53:14Z D! ECS_SD_Stats: Latency: 43.399783ms
```

The meaning of each metric for a particular ECS service discovery cycle is as follows:

- **AWSCLI_DescribeContainerInstances** – the number of `ECS::DescribeContainerInstances` API calls made.
- **AWSCLI_DescribeInstancesRequest** – the number of `ECS::DescribeInstancesRequest` API calls made.
- **AWSCLI_DescribeTaskDefinition** – the number of `ECS::DescribeTaskDefinition` API calls made.
- **AWSCLI_DescribeTasks** – the number of `ECS::DescribeTasks` API calls made.
- **AWSCLI_ListTasks** – the number of `ECS::ListTasks` API calls made.
- **ExporterDiscoveredTargetCount** – the number of Prometheus targets that were discovered and successfully exported into the target result file within the container.
- **LRUCache_Get_EC2MetaData** – the number of times that container instances metadata was retrieved from the cache.
- **LRUCache_Get_TaskDefinition** – the number of times that ECS task definition metadata was retrieved from the cache.
- **LRUCache_Size_ContainerInstance** – the number of unique container instance's metadata cached in memory.
- **LRUCache_Size_TaskDefinition** – the number of unique ECS task definitions cached in memory.
- **Latency** – how long the service discovery cycle takes.

Check the value of `ExporterDiscoveredTargetCount` to see whether the discovered Prometheus targets match your expectations. If not, the possible reasons are as follows:

- The configuration of ECS service discovery might not match your application's setting. For the docker label-based service discovery, your target containers may not have the necessary docker label configured in the CloudWatch agent to auto discover them. For the ECS task definition ARN regular expression-based service discovery, the regex setting in the CloudWatch agent may not match your application's task definition.
- The CloudWatch agent's ECS task role might not have permission to retrieve the metadata of ECS tasks. Check that the CloudWatch agent has been granted the following read-only permissions:

- `ec2:DescribeInstances`
- `ecs>ListTasks`
- `ecs:DescribeContainerInstances`
- `ecs:DescribeTasks`
- `ecs:DescribeTaskDefinition`

Step 3: Check the network connection and the ECS task role policy

If there are still no log events sent to the target CloudWatch Logs log group even though the value of `Exporter_DiscoveredTargetCount` indicates that there are discovered Prometheus targets, this could be caused by one of the following:

- The CloudWatch agent might not be able to connect to the Prometheus target ports. Check the security group setting behind the CloudWatch agent. The private IP should allow the CloudWatch agent to connect to the Prometheus exporter ports.
- The CloudWatch agent's ECS task role might not have the **CloudWatchAgentServerPolicy** managed policy. The CloudWatch agent's ECS task role needs to have this policy to be able to send the Prometheus metrics as log events. If you used the sample AWS CloudFormation template to create the IAM roles automatically, both the ECS task role and the ECS execution role are granted with the least privilege to perform the Prometheus monitoring.

Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters

This section provides help for troubleshooting your Prometheus metrics setup on Amazon EKS and Kubernetes clusters.

General troubleshooting steps on Amazon EKS

To confirm that the CloudWatch agent is running, enter the following command.

```
kubectl get pod -n amazon-cloudwatch
```

The output should include a row with `cwagent-prometheus-id` in the NAME column and Running in the STATUS column.

To display details about the running pod, enter the following command. Replace `pod-name` with the complete name of your pod that has a name that starts with `cw-agent-prometheus`.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

If you have CloudWatch Container Insights installed, you can use CloudWatch Logs Insights to query the logs from the CloudWatch agent collecting the Prometheus metrics.

To query the application logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the log group for the application logs, `/aws/containerinsights/cluster-name/application`
4. Replace the search query expression with the following query, and choose **Run query**

```
fields ispresent(kubernetes.pod_name) as haskubernetes_pod_name, stream,  
kubernetes.pod_name, log |
```

```
filter haskubernetes_pod_name and kubernetes.pod.name like /cwagent-prometheus
```

You can also confirm that Prometheus metrics and metadata are being ingested as CloudWatch Logs events.

To confirm that Prometheus data is being ingested

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the **/aws/containerinsights/*cluster-name*/prometheus**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields @timestamp, @message | sort @timestamp desc | limit 20
```

Logging dropped Prometheus metrics

This release does not collect Prometheus metrics of the histogram type. You can use the CloudWatch agent to check whether any Prometheus metrics are being dropped because they are histogram metrics. You can also log a list of the first 500 Prometheus metrics that are dropped and not sent to CloudWatch because they are histogram metrics.

To see whether any metrics are being dropped, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

If any metrics are being dropped, you will see the following lines in the `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` file.

```
I! Drop Prometheus metrics with unsupported types. Only Gauge, Counter and Summary are supported.  
I! Please enable CWAgent debug mode to view the first 500 dropped metrics
```

If you see those lines and want to know what metrics are being dropped, use the following steps.

To log a list of dropped Prometheus metrics

1. Change the CloudWatch agent to debug mode by adding the following bold lines to your `prometheus-eks.yaml` or `prometheus-k8s.yaml` file, and save the file.

```
{  
  "agent": {  
    "debug": true  
  },
```

This section of the file should then look like this:

```
cwagentconfig.json: |  
{  
  "agent": {  
    "debug": true  
  },  
  "logs": {  
    "metrics_collected": {
```

2. Reinstall the CloudWatch agent to enable debug mode by entering the following commands:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
kubectl apply -f prometheus.yaml
```

The dropped metrics are logged in the CloudWatch agent pod.

3. To retrieve the logs from the CloudWatch agent pod, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

Or, if you have Container Insights FluentD logging installed, the logs are also saved in the CloudWatch Logs log group **/aws/containerinsights/*cluster_name*/application**.

To query these logs, you can follow the steps for querying the application logs in [General troubleshooting steps on Amazon EKS \(p. 451\)](#).

Where are the Prometheus metrics ingested as CloudWatch Logs log events?

The CloudWatch agent creates a log stream for each Prometheus scrape job configuration. For example, in the `prometheus-eks.yaml` and `prometheus-k8s.yaml` files, the line `job_name: 'kubernetes-pod-appmesh-envoy'` scrapes App Mesh metrics. The Prometheus target is defined as `kubernetes-pod-appmesh-envoy`. So all App Mesh Prometheus metrics are ingested as CloudWatch Logs events in the log stream **kubernetes-pod-appmesh-envoy** under the log group named **/aws/containerinsights/*cluster-name*/Prometheus**.

I don't see Amazon EKS or Kubernetes Prometheus metrics in CloudWatch metrics

First, make sure that the Prometheus metrics are ingested as log events in the log group **/aws/containerinsights/*cluster-name*/Prometheus**. Use the information in [Where are the Prometheus metrics ingested as CloudWatch Logs log events? \(p. 453\)](#) to help you check the target log stream. If the log stream is not created or there are no new log events in the log stream, check the following:

- Check that the Prometheus metrics exporter endpoints are set up correctly
- Check that the Prometheus scraping configurations in the `config_map: cwagent-prometheus` section of the CloudWatch agent YAML file is correct. The configuration should be the same as it would be in a Prometheus configuration file. For more information, see `<scrape_config>` in the Prometheus documentation.

If the Prometheus metrics are ingested as log events correctly, check that the embedded metric format settings are added into the log events to generate the CloudWatch metrics.

```
"CloudWatchMetrics": [
    {
        "Metrics": [
            {
                "Name": "envoy_http_downstream_cx_destroy_remote_active_rq"
            }
        ],
        "Dimensions": [
            [
                "ClusterName",
                "Namespace"
            ]
        ],
        "Namespace": "ContainerInsights/Prometheus"
    }
],
```

For more information about embedded metric format, see [Specification: Embedded metric format \(p. 763\)](#).

If there is no embedded metric format in the log events, check that the `metric_definitions` are configured correctly in the `config_map: prometheus-cwagentconfig` section of the CloudWatch agent installation YAML file. For more information, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics \(p. 422\)](#).

Integration with Application Insights

Amazon CloudWatch Application Insights helps you monitor your applications and identifies and sets up key metrics, logs, and alarms across your application resources and technology stack. For more information, see [Amazon CloudWatch Application Insights \(p. 603\)](#).

You can enable Application Insights to gather additional data from your containerized applications and microservices. If you haven't done this already, you can enable it by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.

If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard.

For more information about Application Insights and containerized applications, see [Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring \(p. 620\)](#).

Troubleshooting Container Insights

The following sections can help if you're having trouble issues with Container Insights.

Failed deployment on Amazon EKS or Kubernetes

If the agent doesn't deploy correctly on a Kubernetes cluster, try the following:

- Run the following command to get the list of pods.

```
kubectl get pods -n amazon-cloudwatch
```

- Run the following command and check the events at the bottom of the output.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- Run the following command to check the logs.

```
kubectl logs pod-name -n amazon-cloudwatch
```

Unauthorized panic: Cannot retrieve cAdvisor data from kubelet

If your deployment fails with the error `Unauthorized panic: Cannot retrieve cAdvisor data from kubelet`, your kubelet might not have Webhook authorization mode enabled. This mode is required for Container Insights. For more information, see [Verify prerequisites \(p. 320\)](#).

Deploying Container Insights on a deleted and re-created cluster on Amazon ECS

If you delete an existing Amazon ECS cluster that does not have Container Insights enabled, and you re-create it with the same name, you can't enable Container Insights on this new cluster at the time you re-create it. You can enable it by re-creating it, and then entering the following command:

```
aws ecs update-cluster-settings --cluster myCICluster --settings  
name=containerInsights,value=enabled
```

Invalid endpoint error

If you see an error message similar to the following, check to make sure that you replaced all the placeholders such as *cluster-name* and *region-name* in the commands that you are using with the correct information for your deployment.

```
"log": "2020-04-02T08:36:16Z E! cloudwatchlogs: code: InvalidEndpointURL, message:  
invalid endpoint uri, original error: &url.Error{Op:\"parse\", URL:\"https://logs.  
{region_name}.amazonaws.com/\", Err:\"{}\", &awserr.baseError{code:\"InvalidEndpointURL  
\", message:\"invalid endpoint uri\", errs:[]error{(*url.Error)(0xc0008723c0)}}\n",
```

Metrics don't appear in the console

If you don't see any Container Insights metrics in the AWS Management Console, be sure that you have completed the setup of Container Insights. Metrics don't appear before Container Insights has been set up completely. For more information, see [Setting up Container Insights \(p. 307\)](#).

Pod metrics missing on Amazon EKS or Kubernetes after upgrading cluster

This section might be useful if you all or some pod metrics are missing after you deploy the CloudWatch agent as a daemonset on a new or upgraded cluster, or you see an error log with the message W! No pod metric collected.

These errors can be caused by changes in the container runtime, such as containerd or the docker systemd cgroup driver. You can usually solve this by updating your deployment manifest so that the containerd socket from the host is mounted into the container. See the following example:

```
# For full example see https://github.com/aws-samples/amazon-cloudwatch-container-insights/  
blob/master/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-  
monitoring/cwagent/cwagent-daemonset.yaml  
apiVersion: apps/v1  
kind: DaemonSet  
metadata:  
  name: cloudwatch-agent  
  namespace: amazon-cloudwatch  
spec:  
  template:  
    spec:  
      containers:  
        - name: cloudwatch-agent  
# ...  
      # Don't change the mountPath  
      volumeMounts:
```

```
# ...
    - name: dockersock
      mountPath: /var/run/docker.sock
      readOnly: true
    - name: varlibdocker
      mountPath: /var/lib/docker
      readOnly: true
    - name: containerdsock # NEW mount
      mountPath: /run/containerd/containerd.sock
      readOnly: true
# ...
    volumes:
# ...
    - name: dockersock
      hostPath:
        path: /var/run/docker.sock
    - name: varlibdocker
      hostPath:
        path: /var/lib/docker
    - name: containerdsock # NEW volume
      hostPath:
        path: /run/containerd/containerd.sock
```

No pod metrics when using Bottlerocket for Amazon EKS

Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers.

Bottlerocket uses a different `containerd` path on the host, so you need to change the volumes to its location. If you don't, you see an error in the logs that includes `W! No pod metric collected`. See the following example.

```
volumes:
# ...
    - name: containerdsock
      hostPath:
        # path: /run/containerd/containerd.sock
        # bottlerocket does not mount containerd sock at normal place
        # https://github.com/bottlerocket-os/bottlerocket/
commit/91810c85b83ff4c3660b496e243ef8b55df0973b
      path: /run/dockershim.sock
```

No container filesystem metrics when using the containerd runtime for Amazon EKS or Kubernetes

This is a known issue and is being worked on by community contributors. For more information, see [Disk usage metric for containerd](#) and [Container file system metrics is not supported by cAdvisor for containerd](#) on GitHub.

Unexpected log volume increase from CloudWatch agent when collecting Prometheus metrics

This was a regression introduced in version 1.247347.6b250880 of the CloudWatch agent. This regression has already been fixed in more recent versions of the agent. Its impact was limited to

scenarios where customers collected the logs of the CloudWatch agent itself and were also using Prometheus. For more information, see [\[prometheus\] agent is printing all the scraped metrics in log on GitHub](#).

Latest docker image mentioned in release notes not found from Dockerhub

We update the release note and tag on Github before we start the actual release internally. It usually takes 1-2 weeks to see the latest docker image on registries after we bump the version number on Github. There is no nightly release for the CloudWatch agent container image. You can build the image directly from source at the following location: <https://github.com/aws/amazon-cloudwatch-agent/tree/master/amazon-cloudwatch-container-insights/cloudwatch-agent-dockerfile>

CrashLoopBackoff error on the CloudWatch agent

If you see a CrashLoopBackOff error for the CloudWatch agent, make sure that your IAM permissions are set correctly. For more information, see [Verify prerequisites \(p. 320\)](#).

CloudWatch agent or FluentD pod stuck in pending

If you have a CloudWatch agent or FluentD pod stuck in Pending or with a FailedScheduling error, determine if your nodes have enough compute resources based on the number of cores and amount of RAM required by the agents. Enter the following command to describe the pod:

```
kubectl describe pod cloudwatch-agent-85ppg -n amazon-cloudwatch
```

Building your own CloudWatch agent Docker image

You can build your own CloudWatch agent Docker image by referring to the Dockerfile located at <https://github.com/aws-samples/amazon-cloudwatch-container-insights/blob/master/cloudwatch-agent-dockerfile/Dockerfile>.

The Dockerfile supports building multi-architecture images directly using `docker buildx`.

Deploying other CloudWatch agent features in your containers

You can deploy additional monitoring features in your containers using the CloudWatch agent. These features include the following:

- **Embedded Metric Format**— For more information, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).
- **StatsD**— For more information, see [Retrieve custom metrics with StatsD \(p. 562\)](#).

Instructions and necessary files are located on GitHub at the following locations:

- For Amazon ECS containers, see [Example Amazon ECS task definitions based on deployment modes](#).
- For Amazon EKS and Kubernetes containers, see [Example Kubernetes YAML files based on deployment modes](#).

Using Lambda Insights

CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. The solution collects, aggregates, and summarizes system-level metrics including CPU time, memory, disk, and network. It also collects, aggregates, and summarizes diagnostic information such as cold starts and Lambda worker shutdowns to help you isolate issues with your Lambda functions and resolve them quickly.

Lambda Insights uses a new CloudWatch Lambda extension, which is provided as a Lambda layer. When you install this extension on a Lambda function, it collects system-level metrics and emits a single performance log event for every invocation of that Lambda function. CloudWatch uses embedded metric formatting to extract metrics from the log events.

For more information about Lambda extensions, see [Using AWS Lambda extensions](#). For more information about embedded metric format, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

You can use Lambda Insights with any Lambda function that uses a Lambda runtime that supports Lambda extensions. For a list of these runtimes, see [Lambda Extensions API](#).

Pricing

For each Lambda function enabled for Lambda Insights, you only pay for what you use for metrics and logs. For a pricing example, see [Amazon CloudWatch Pricing](#).

You are charged for the execution time consumed by the Lambda extension, in 1ms increments. For more information about Lambda pricing, see [AWS Lambda Pricing](#).

Getting started with Lambda Insights

To enable Lambda Insights on a Lambda function, you can use a one-click toggle in the Lambda console. Alternatively, you can use the AWS CLI, AWS CloudFormation, the AWS Serverless Application Model CLI, or the AWS Cloud Development Kit (CDK).

The following sections provide detailed instructions for completing these steps.

Topics

- [Available versions of the Lambda Insights extension \(p. 459\)](#)
- [Using the console to enable Lambda Insights on an existing Lambda function \(p. 469\)](#)
- [Using the AWS CLI to enable Lambda Insights on an existing Lambda function \(p. 469\)](#)
- [Using the AWS SAM CLI to enable Lambda Insights on an existing Lambda function \(p. 470\)](#)
- [Using AWS CloudFormation to enable Lambda Insights on an existing Lambda function \(p. 471\)](#)
- [Using the AWS CDK to enable Lambda Insights on an existing Lambda function \(p. 472\)](#)
- [Using Serverless Framework to enable Lambda Insights on an existing Lambda function \(p. 473\)](#)
- [Enabling Lambda Insights on a Lambda container image deployment \(p. 474\)](#)

Available versions of the Lambda Insights extension

This section lists the versions of the Lambda Insights extension, and the ARNs to use for these extensions in each AWS Region.

Topics

- [x86-64 platforms \(p. 460\)](#)
- [ARM64 platforms \(p. 467\)](#)

x86-64 platforms

This section lists the versions of the Lambda Insights extension for x86-64 platforms, and the ARNs to use for these extensions in each AWS Region.

1.0.135.0

Version 1.0.135.0 includes bug fixes for how Lambda Insights collects and reports disk and file descriptor usage. In previous versions of the extension, the `tmp_free` metric reported the maximum free space in the `/tmp` directory while a function runs. This version changes the metric to report the minimum value instead, making it more useful when assessing disk usage. For more information about `tmp` directory storage quotas, see [Lambda quotas](#).

Version 1.0.135.0 also now reports file descriptor usage (`fd_use` and `fd_max`) as the maximum value across processes rather than reporting the operating system level.

ARNs for version 1.0.135.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:18</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:18</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:18</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:18</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:25</code>

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:18
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:11
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:11
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:18
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:18
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:18
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:11
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:18
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:18
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:11
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:18

1.0.119.0

ARNs for version 1.0.119.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:16
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:16
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:16
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:16
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:9

Region	ARN
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:9
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:23
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:16
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:9
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:9
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:16
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:16
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:16
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:9
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:16
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:16
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:9
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:16

1.0.98.0

This version removes unnecessary logging and also addresses an issue with the AWS Serverless Application Model CLI local invokes. For more information about this issue, see [Adding LambdaInsightsExtension results in timeout with 'sam local invoke'](#).

ARNs for version 1.0.98.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:14</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:14</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:14</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:8</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:8</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:14</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:14</code>
China (Beijing)	<code>arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:8</code>
China (Ningxia);	<code>arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:8</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:14</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:14</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:14</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:8</code>

Region	ARN
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:14</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:14</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:8</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:14</code>

1.0.89.0

This version corrects the performance event timestamp to always represent the start of the invocation of the function.

ARNs for version 1.0.89.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:12</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:12</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:12</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:12</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:12</code>

Region	ARN
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:12</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:12</code>

1.0.86.0

With version 1.0.54.0 of the extension, memory metrics were sometimes reported incorrectly and sometimes were higher than 100%. Version 1.0.86.0 corrects the memory measurement issue by using the same event data as Lambda platform metrics. This means that you may see a dramatic change in the recorded memory metric values. This is achieved by using the new Lambda Logs API. This provides a more accurate measurement of Lambda sandbox memory usage. However, something to be aware of is that the Lambda Logs API can't deliver platform report events if a function sandbox times out and is subsequently spun down. In this case, Lambda Insights is unable to record the invocation metrics. For more information about Lambda Logs API, see [AWS Lambda Logs API](#).

New features in version 1.0.86.0

- Uses the Lambda Logs API to correct the memory metric. This solves the previous issue where memory statistics were greater than 100%.
- Introduces `Init Duration` as a new CloudWatch metric.
- Uses the invocation ARN to add a `version` dimension for aliases and invoked versions. If you are using Lambda aliases or versions to achieve incremental deployments (such as blue-green deployments), you can view your metrics based on the invoked alias. The `version` dimension is not applied if the function doesn't use an alias or version. For more information, see [Lambda function aliases](#).
- Adds a `billed_mb_ms` field to the performance events to display the cost per invoke. This does not consider any cost associated with provisioned concurrency.
- Adds `billed_duration` and `duration` fields to the performance events.

ARNs for version 1.0.86.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:11</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:11</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:11</code>

Region	ARN
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:11</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:11</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:11</code>

1.0.54.0

Version 1.0.54.0 was the initial release of the Lambda Insights extension.

ARNs for version 1.0.54.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:2</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:2</code>

Region	ARN
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:2
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:2
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:2
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:2
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:2
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:2
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:2
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:2
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:2

ARM64 platforms

This section lists the versions of the Lambda Insights extension for ARM64 platforms, and the ARNs to use for these extensions in each AWS Region.

1.0.135.0

Version 1.0.135.0 includes bug fixes for how Lambda Insights collects and reports disk and file descriptor usage. In previous versions of the extension, the `tmp_free` metric reported the maximum free space in the `/tmp` directory while a function runs. This version changes the metric to report the minimum value instead, making it more useful when assessing disk usage. For more information about `tmp` directory storage quotas, see [Lambda quotas](#).

Version 1.0.135.0 also now reports file descriptor usage (`fd_use` and `fd_max`) as the maximum value across processes rather than reporting the operating system level.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>

1.0.119.0

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>

Region	ARN
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1</code>

Using the console to enable Lambda Insights on an existing Lambda function

Use the following steps in the Lambda Console to enable Lambda Insights on an existing Lambda function.

To enable Lambda Insights on a Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose the name of a function, and then select the **Configuration** tab on the following screen.

After you choose the name of a function, you're directed to a screen that contains an overview of the function.

3. Under the **Configuration** tab, choose **Monitoring tools** in the left navigation menu, and then choose **Edit**.
You're directed to a screen where you can edit monitoring tools.
4. Under the section **CloudWatch Lambda Insights**, enable **Enhanced monitoring**, and then choose **Save**.

Using the AWS CLI to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS CLI to enable Lambda Insights on an existing Lambda function.

Step 1: Update function permissions

To update the function's permissions

- Attach the **CloudWatchLambdaInsightsExecutionRolePolicy** managed IAM policy to the function's execution role by entering the following command.

```
aws iam attach-role-policy \
```

```
--role-name function-execution-role \  
--policy-arn "arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy"
```

Step 2: Install the Lambda extension

Install the Lambda extension by entering the following command. Replace the ARN value for the `layers` parameter with the ARN matching your Region and the extension version that you want to use. For more information, see [Available versions of the Lambda Insights extension \(p. 459\)](#).

```
aws lambda update-function-configuration \  
--function-name function-name \  
--layers "arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14"
```

Step 3: Enable the CloudWatch Logs VPC endpoint

This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs virtual private cloud (VPC) endpoint.

If you need to do this step, enter the following command, replacing the placeholders with information for your VPC.

For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#).

```
aws ec2 create-vpc-endpoint \  
--vpc-id vpcId \  
--vpc-endpoint-type Interface \  
--service-name com.amazonaws.region.logs \  
--subnet-id subnetId \  
--security-group-id securitygroupId
```

Using the AWS SAM CLI to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS SAM AWS CLI to enable Lambda Insights on an existing Lambda function.

If you don't already have the latest version of the AWS SAM CLI installed, you must first install or upgrade it. For more information, see [Installing the AWS SAM CLI](#).

Step 1: Install the layer

To make the Lambda Insights extension available to all of your Lambda functions, add a `Layers` property to the `Globals` section of your SAM template with the ARN of the Lambda Insights layer. The example below uses the layer for the initial release of Lambda Insights. For the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension \(p. 459\)](#).

```
Globals:  
  Function:  
    Layers:  
      - !Sub "arn:aws:lambda:${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

To enable this layer for only a single function, add the `Layers` property to the function as shown in this example.

```
Resources:
```

```
MyFunction:  
  Type: AWS::Serverless::Function  
  Properties:  
    Layers:  
      - !Sub "arn:aws:lambda:  
        ${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

Step 2: Add the managed policy

For each function, add the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy.

AWS SAM doesn't support global policies, so you must enable those on each function individually, as shown in this example. For more information about globals, see [Globals Section](#).

```
Resources:  
  MyFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      Policies:  
        - CloudWatchLambdaInsightsExecutionRolePolicy
```

Invoking locally

The AWS SAM CLI supports Lambda extensions. However, every locally executed invocation resets the runtime environment. Lambda Insights data won't be available from local invocations because the runtime is restarted without a shutdown event. For more information, see [Release 1.6.0 - Add support for local testing of AWS Lambda extensions](#).

Troubleshooting

To troubleshoot your Lambda Insights installation, add the following environment variable to your Lambda function to enable debug logging.

```
Resources:  
  MyFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      Environment:  
        Variables:  
          LAMBDA_INSIGHTS_LOG_LEVEL: info
```

Using AWS CloudFormation to enable Lambda Insights on an existing Lambda function

Follow these steps to use AWS CloudFormation to enable Lambda Insights on an existing Lambda function.

Step 1: Install the layer

Add the Lambda Insights layer to the `Layers` property within the Lambda Insights layer ARN. The example below uses the layer for the initial release of Lambda Insights. For the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension \(p. 459\)](#).

```
Resources:  
  MyFunction:  
    Type: AWS::Lambda::Function  
    Properties:  
      Layers:
```

```
- !Sub "arn:aws:lambda:  
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

Step 2: Add the managed policy

Add the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to your function execution role.

```
Resources:  
  MyFunctionExecutionRole:  
    Type: 'AWS::IAM::Role'  
    Properties:  
      ManagedPolicyArns:  
        - 'arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy'
```

Step 3: (Optional) Add VPC endpoint

This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs virtual private cloud (VPC) endpoint. For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#).

```
Resources:  
  CloudWatchLogsVpcPrivateEndpoint:  
    Type: AWS::EC2::VPCEndpoint  
    Properties:  
      PrivateDnsEnabled: 'true'  
      VpcEndpointType: Interface  
      VpcId: !Ref VPC  
      ServiceName: !Sub com.amazonaws.${AWS::Region}.logs  
      SecurityGroupIds:  
        - !Ref InterfaceVpcEndpointSecurityGroup  
      SubnetIds:  
        - !Ref PublicSubnet01  
        - !Ref PublicSubnet02  
        - !Ref PublicSubnet03
```

Using the AWS CDK to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS CDK to enable Lambda Insights on an existing Lambda function. To use these steps, you must already be using the AWS CDK to manage your resources.

The commands in this section are in TypeScript.

First, update the function permissions.

```
executionRole.addManagedPolicy(  
  ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaInsightsExecutionRolePolicy'));
```

Next, install the extension on the Lambda function. Replace the ARN value for the `layerArn` parameter with the ARN matching your Region and the extension version that you want to use. For more information, see [Available versions of the Lambda Insights extension \(p. 459\)](#).

```
import lambda = require('@aws-cdk/aws-lambda');  
const layerArn = `arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14`;  
const layer = lambda.LayerVersion.fromLayerVersionArn(this, `LayerFromArn`, layerArn);
```

If necessary, enable the virtual private cloud (VPC) endpoint for CloudWatch Logs. This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs VPC endpoint.

```
const cloudWatchLogsEndpoint = vpc.addInterfaceEndpoint('cwl-gateway', {  
    service: InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,  
});  
  
cloudWatchLogsEndpoint.connections.allowDefaultPortFromAnyIpv4();
```

Using Serverless Framework to enable Lambda Insights on an existing Lambda function

Follow these steps to use Serverless Framework to enable Lambda Insights on an existing Lambda function. For more information about Serverless Framework, see [serverless.com](#).

This is done through a Lambda Insights plugin for Serverless. For more information, see [serverless-plugin-lambda-insights](#).

If you don't already have the latest version of the Serverless command-line interface installed, you must first install or upgrade it. For more information, see [Get started with Serverless Framework Open Source & AWS](#).

To use Serverless Framework to enable Lambda Insights on a Lambda function

1. Install the Serverless plugin for Lambda Insights by running the following command in your Serverless directory:

```
npm install --save-dev serverless-plugin-lambda-insights
```

2. In your `serverless.yml` file, add the plugin in the `plugins` section as shown:

```
provider:  
  name: aws  
plugins:  
  - serverless-plugin-lambda-insights
```

3. Enable Lambda Insights.

- You can enable Lambda Insights individually per function by adding the following property to the `serverless.yml` file

```
functions:  
  myLambdaFunction:  
    handler: src/app/index.handler  
    lambdaInsights: true #enables Lambda Insights for this function
```

- You can enable Lambda Insights for all functions within the `serverless.yml` file by adding the following custom section:

```
custom:  
  lambdaInsights:  
    defaultLambdaInsights: true #enables Lambda Insights for all functions
```

4. Re-deploy the Serverless service by entering the following command:

```
serverless deploy
```

This re-deploys all functions and enables Lambda Insights for those functions that you have specified. It enables Lambda Insights by adding the Lambda Insights layer and attaching the necessary permissions using the `arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy` IAM policy.

Enabling Lambda Insights on a Lambda container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image, add the following lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image.

```
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm
```

After you create your Lambda function, assign the `CloudWatchLambdaInsightsExecutionRolePolicy` IAM policy to the function's execution role, and Lambda Insights is enabled on the container image-based Lambda function.

Note

To use an older version of the Lambda Insights extension, replace the URL in the previous commands with this URL: `https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension.1.0.111.0.rpm`. Currently, only Lambda Insights versions 1.0.111.0 and later are available. For more information, see [Available versions of the Lambda Insights extension \(p. 459\)](#).

To verify the signature of the Lambda Insights agent package on a Linux server

1. Enter the following command to download the public key.

```
shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/lambda-insights-extension.gpg
```

2. Enter the following command to import the public key into your keyring.

```
shell$ gpg --import lambda-insights-extension.gpg
```

The output will be similar to the following. Make a note of the key value, you will need it in the next step. In this example output, the key value is 848ABDC8.

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

3. Verify the fingerprint by entering the following command. Replace `key-value` with the value of the key from the preceding step.

```
shell$ gpg --fingerprint key-value
```

The fingerprint string in the output of this command should be `E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8`. If the string doesn't match, don't install the agent and contact AWS.

4. After you have verified the fingerprint, you can use it to verify the Lambda Insights agent package. Download the package signature file by entering the following command.

```
shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/  
amazon_linux/lambda-insights-extension.rpm.sig
```

5. Verify the signature by entering the following command.

```
shell$ gpg --verify lambda-insights-extension.rpm.sig lambda-insights-extension.rpm
```

The output should look like the following:

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8  
gpg: Good signature from "Amazon Lambda Insights Extension"  
gpg: WARNING: This key is not certified with a trusted signature!  
gpg: There is no indication that the signature belongs to the owner.  
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8
```

In the expected output, there might be a warning about a trusted signature. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

If the output contains BAD signature, check whether you performed the steps correctly. If you continue to get a BAD signature response, contact AWS and avoid using the downloaded file.

Example

This section includes an example of enabling Lambda Insights on a container image-based Python Lambda function.

An example of enabling Lambda Insights on a Lambda container image

1. Create a Dockerfile that is similar to the following:

```
FROM public.ecr.aws/lambda/python:3.8

// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/  
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. Create a Python file named `index.py` that is similar to the following:

```
def handler(event, context):
    return {
        'message': 'Hello World!'
    }
```

3. Put the Dockerfile and `index.py` in the same directory. Then, in that directory, run the following steps to build the docker image and upload it to Amazon ECR.

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
```

```
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
"${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-repository:latest
```

4. Use that Amazon ECR image that you just created to create the Lambda function.
5. Assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role.

Viewing your Lambda Insights metrics

After you have installed the Lambda Insights extension on a Lambda function that has been invoked, you can use the CloudWatch console to see your metrics. You can see a multi-function overview, or focus on a single function.

For a list of Lambda Insights metrics, see [Metrics collected by Lambda Insights \(p. 476\)](#).

To view the multi-function overview for your Lambda Insights metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Lambda Insights**, choose **Multi-function**.

The top part of the page displays graphs with aggregated metrics of all your Lambda functions in the Region that have Lambda Insights enabled. Lower on the page is a table that lists the functions.

3. To filter by function name to reduce the number of functions displayed, type part of the function name in the box near the top of the page.
4. To add this view to a dashboard as a widget, choose **Add to dashboard**.

To view metrics for a single function

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Lambda Insights**, choose **Single-function**.

The top part of the page displays graphs with metrics for the selected function.

3. If you have X-Ray enabled, you can choose a single trace ID. This opens CloudWatch ServiceLens for that invocation, and from there you can zoom out to see the distributed trace and the other services involved in handling that specific transaction. For more information about ServiceLens, see [Using ServiceLens to monitor the health of your applications \(p. 268\)](#).
4. To open CloudWatch Logs Insights and zoom in on a specific error, choose **View logs** by the table at the bottom of the page.
5. To add this view to a dashboard as a widget, choose **Add to dashboard**.

Metrics collected by Lambda Insights

Lambda Insights collects several metrics from the Lambda functions where it is installed. Some of these metrics are available as time series aggregated data in CloudWatch Metrics. Other metrics are not

aggregated into time series data but can be found in the embedded metric format log entries by using CloudWatch Logs Insights.

The following metrics are available as time series aggregated data in CloudWatch Metrics in the LambdaInsights namespace.

Metric name	Dimensions	Description
cpu_total_time	function_name function_name, version	Sum of cpu_system_time and cpu_user_time. Unit: Milliseconds
init_duration	function_name function_name, version	The amount of time spent in the init phase of the Lambda execution environment lifecycle. Unit: Milliseconds
memory_utilization	function_name function_name, version	The maximum memory measured as a percentage of the memory allocated to the function. Unit: Percent
rx_bytes	function_name function_name, version	The number of bytes received by the function. Unit: Bytes
tx_bytes	function_name function_name, version	The number of bytes sent by the function. Unit: Bytes
total_memory	function_name function_name, version	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes
total_network	function_name function_name, version	Sum of rx_bytes and tx_bytes. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes

Metric name	Dimensions	Description
used_memory_max	function_name function_name, version	The measured memory of the function sandbox. Unit: Megabytes

The following metrics can be found in the embedded metric format log entries by using CloudWatch Logs Insights. For more information about CloudWatch Logs Insights, see [Analyzing Log Data with CloudWatch Logs Insights](#).

For more information about embedded metric format, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

Metric name	Description
cpu_system_time	The amount of time the CPU spent executing kernel code. Unit: Milliseconds
cpu_total_time	Sum of <code>cpu_system_time</code> and <code>cpu_user_time</code> . Unit: Milliseconds
cpu_user_time	The amount of time the CPU spent executing user code. Unit: Milliseconds
fd_max	The maximum number of file descriptors available. Unit: Count
fd_use	The maximum number of file descriptors in use. Unit: Count
memory_utilization	The maximum memory measured as a percentage of the memory allocated to the function. Unit: Percent
rx_bytes	The number of bytes received by the function. Unit: Bytes
tx_bytes	The number of bytes sent by the function. Unit: Bytes
threads_max	The number of threads in use by the function process. As a function author, you don't control the initial number of threads created by the runtime. Unit: Count

Metric name	Description	
tmp_max	The amount of space available in the /tmp directory. Unit: Bytes	
tmp_used	The amount of space used in the /tmp directory. Unit: Bytes	
total_memory	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes	
total_network	Sum of rx_bytes and tx_bytes. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes	
used_memory_max	The measured memory of the function sandbox. Unit: Bytes	

Troubleshooting and known issues

The first step to troubleshooting any issues is to enable debug logging on the Lambda Insights extension. To do this, set the following environment variable on your Lambda function:

`LAMBDA_INSIGHTS_LOG_LEVEL=info`. For more information, see [Using AWS Lambda environment variables](#).

The extension emits logs into the same log group as your function (`/aws/lambda/function-name`). Review those logs to see if the error might be related to a setup issue.

I don't see any metrics from Lambda Insights

If you don't see Lambda Insights metrics that you expect to see, check the following possibilities:

- **The metrics might just be delayed**—If the function has not yet been invoked or the data has not been flushed yet, you won't see the metrics in CloudWatch. For more information, see [Known Issues](#) later in this section.
- **Confirm that the Lambda function has the correct permissions**—Make sure that the `CloudWatchLambdaInsightsExecutionRolePolicy` IAM policy is assigned to the function's execution role.
- **Check the Lambda runtime**—Lambda Insights supports only certain Lambda runtimes. For a list of supported runtimes, see [Using Lambda Insights \(p. 459\)](#).

For example, to use Lambda Insights on Java 8, you must use the `java8.al2` runtime, not the `java8` runtime.

- **Check network access**—The Lambda function might be on a VPC private subnet with no internet access and you don't have a VPC endpoint configured for CloudWatch Logs. To help debug this issue, you can set the environment variable `LAMBDA_INSIGHTS_LOG_LEVEL=info`.

Known issues

Data delay can be as high as 20 minutes. When a function handler completes, Lambda freezes the sandbox, which also freezes the Lambda Insights extension. While the function is running, we use an adaptive batching strategy based on the function TPS to output data. However, if the function stops being invoked for an extended period and there is still event data in the buffer, this data can be delayed until Lambda shuts down the idle sandbox. When Lambda shuts down the sandbox, we flush the buffered data.

Example telemetry event

Each invocation of a Lambda function that has Lambda Insights enabled writes a single log event to the `/aws/lambda-insights` log group. Each log event contains metrics in embedded metric format. For more information about embedded metric format, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

To analyze these log events, you can use the following methods:

- The Lambda Insights section of the CloudWatch console, as explained in [Viewing your Lambda Insights metrics \(p. 476\)](#).
- Log event queries using CloudWatch Logs Insights. For more information, see [Analyzing Log Data with CloudWatch Logs Insights](#).
- Metrics collected in the `LambdaInsights` namespace, which you graph by using CloudWatch metrics.

The following is an example of a Lambda Insights log event with embedded metric format.

```
{  
  "_aws": {  
    "Timestamp": 1605034324256,  
    "CloudWatchMetrics": [  
      {  
        "Namespace": "LambdaInsights",  
        "Dimensions": [  
          [ "function_name" ],  
          [ "function_name", "version" ]  
        ],  
        "Metrics": [  
          { "Name": "memory_utilization", "Unit": "Percent" },  
          { "Name": "total_memory", "Unit": "Megabytes" },  
          { "Name": "used_memory_max", "Unit": "Megabytes" },  
          { "Name": "cpu_total_time", "Unit": "Milliseconds" },  
          { "Name": "tx_bytes", "Unit": "Bytes" },  
          { "Name": "rx_bytes", "Unit": "Bytes" },  
          { "Name": "total_network", "Unit": "Bytes" },  
          { "Name": "init_duration", "Unit": "Milliseconds" }  
        ]  
      }  
    ],  
    "LambdaInsights": {  
      "ShareTelemetry": true  
    }  
  },  
  "event_type": "performance",  
  "function_name": "cpu-intensive",  
  "version": "Blue",  
  "request_id": "12345678-8bcc-42f7-b1de-123456789012",  
  "trace_id": "1-5faae118-12345678901234567890",  
}
```

```
"duration": 45191,  
"billed_duration": 45200,  
"billed_mb_ms": 11571200,  
"cold_start": true,  
"init_duration": 130,  
"tmp_free": 538329088,  
"tmp_max": 551346176,  
"threads_max": 11,  
"used_memory_max": 63,  
"total_memory": 256,  
"memory_utilization": 24,  
"cpu_user_time": 6640,  
"cpu_system_time": 50,  
"cpu_total_time": 6690,  
"fd_use": 416,  
"fd_max": 32642,  
"tx_bytes": 4434,  
"rx_bytes": 6911,  
"timeout": true,  
"shutdown_reason": "Timeout",  
"total_network": 11345,  
"agent_version": "1.0.72.0",  
"agent_memory_avg": 10,  
"agent_memory_max": 10  
}
```

Collecting metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent

The unified CloudWatch agent enables you to do the following:

- Collect internal system-level metrics from Amazon EC2 instances across operating systems. The metrics can include in-guest metrics, in addition to the metrics for EC2 instances. The additional metrics that can be collected are listed in [Metrics collected by the CloudWatch agent \(p. 574\)](#).
- Collect system-level metrics from on-premises servers. These can include servers in a hybrid environment as well as servers not managed by AWS.
- Retrieve custom metrics from your applications or services using the `StatsD` and `collectd` protocols. `StatsD` is supported on both Linux servers and servers running Windows Server. `collectd` is supported only on Linux servers.
- Collect logs from Amazon EC2 instances and on-premises servers, running either Linux or Windows Server.

Note

The CloudWatch agent does not support collecting logs from FIFO pipes.

You can store and view the metrics that you collect with the CloudWatch agent in CloudWatch just as you can with any other CloudWatch metrics. The default namespace for metrics collected by the CloudWatch agent is `CWAgent`, although you can specify a different namespace when you configure the agent.

The logs collected by the unified CloudWatch agent are processed and stored in Amazon CloudWatch Logs, just like logs collected by the older CloudWatch Logs agent. For information about CloudWatch Logs pricing, see [Amazon CloudWatch Pricing](#).

Metrics collected by the CloudWatch agent are billed as custom metrics. For more information about CloudWatch metrics pricing, see [Amazon CloudWatch Pricing](#).

The CloudWatch agent is open-source under the MIT license, and is [hosted on GitHub](#). If you would like to build, customize or contribute to the CloudWatch agent, see the GitHub repository for the latest instructions. If you think you've found a potential security issue, do not post it on GitHub or any public forum. Instead, please follow the instructions at [Vulnerability Reporting](#) or [email AWS security directly](#).

The steps in this section explain how to install the unified CloudWatch agent on Amazon EC2 instances and on-premises servers. For more information about the metrics that the CloudWatch agent can collect, see [Metrics collected by the CloudWatch agent \(p. 574\)](#).

Supported operating systems

The CloudWatch agent is supported on x86-64 architecture on the following operating systems:

- Amazon Linux version 2014.03.02 or later
- Amazon Linux 2
- Ubuntu Server versions 20.04, 18.04, 16.04, and 14.04
- CentOS versions 8.0, 7.6, 7.2, and 7.0
- Red Hat Enterprise Linux (RHEL) versions 8, 7.7, 7.6, 7.5, 7.4, 7.2, and 7.0
- Debian version 10 and version 8.0
- SUSE Linux Enterprise Server (SLES) version 15 and version 12
- Oracle Linux versions 7.8, 7.6, and 7.5
- macOS, including EC2 Mac1 instances
- 64-bit versions of Windows Server 2019, Windows Server 2016, and Windows Server 2012

The agent is supported on ARM64 architecture on the following operating systems:

- Amazon Linux 2
- Ubuntu Server versions 20.04 and 18.04
- Red Hat Enterprise Linux (RHEL) version 7.6
- SUSE Linux Enterprise Server 15

Installation process overview

You can download and install the CloudWatch agent manually using the command line, or you can integrate it with SSM. The general flow of installing the CloudWatch agent using either method is as follows:

1. Create IAM roles or users that enable the agent to collect metrics from the server and optionally to integrate with AWS Systems Manager.
2. Download the agent package.
3. Modify the CloudWatch agent configuration file and specify the metrics that you want to collect.
4. Install and start the agent on your servers. As you install the agent on an EC2 instance, you attach the IAM role that you created in step 1. As you install the agent on an on-premises server, you specify a named profile that contains the credentials of the IAM user that you created in step 1.

Contents

- [Installing the CloudWatch agent \(p. 483\)](#)
- [Create the CloudWatch agent configuration file \(p. 522\)](#)
- [Metrics collected by the CloudWatch agent \(p. 574\)](#)
- [OpenTelemetry support in the CloudWatch agent \(p. 584\)](#)
- [Common scenarios with the CloudWatch agent \(p. 589\)](#)
- [Troubleshooting the CloudWatch agent \(p. 596\)](#)

Installing the CloudWatch agent

The CloudWatch agent is available as a package in Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances \(p. 499\)](#).

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems, you can download and install the CloudWatch agent using either the command line with an Amazon S3 download link, using SSM, or using an AWS CloudFormation template.

Contents

- [Installing the CloudWatch agent using the command line \(p. 484\)](#)
- [Installing the CloudWatch agent using AWS Systems Manager \(p. 498\)](#)
- [Installing the CloudWatch agent on new instances using AWS CloudFormation \(p. 511\)](#)
- [Verifying the signature of the CloudWatch agent package \(p. 515\)](#)

Installing the CloudWatch agent using the command line

Use the following topics to download, configure, and install the CloudWatch agent package.

Topics

- [Download and configure the CloudWatch agent using the command line \(p. 484\)](#)
- [Create IAM roles and users for use with CloudWatch agent \(p. 489\)](#)
- [Installing and running the CloudWatch agent on your servers \(p. 491\)](#)

Download and configure the CloudWatch agent using the command line

Use the following steps to download the CloudWatch agent package, create IAM roles or users, and optionally modify the common configuration file.

Download the CloudWatch agent package

The CloudWatch agent is available as a package in Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles and users for use with CloudWatch agent \(p. 489\)](#).

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems, you can download and install the CloudWatch agent using the command line.

For each download link, there is a general link as well as links for each Region. For example, for Amazon Linux and Amazon Linux 2 and the x86-64 architecture, three of the valid download links are:

- https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://s3.us-east-1.amazonaws.com/amazoncloudwatch-agent-us-east-1/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://s3.eu-central-1.amazonaws.com/amazoncloudwatch-agent-eu-central-1/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

You can also download a README file about the latest changes to the agent, and a file that indicates the version number that is available for download. These files are in the following locations:

- https://s3.amazonaws.com/amazoncloudwatch-agent/info/latest/RELEASE_NOTES or
https://s3.region.amazonaws.com/amazoncloudwatch-agent-Region/info/latest/RELEASE_NOTES
- https://s3.amazonaws.com/amazoncloudwatch-agent/info/latest/CWAGENT_VERSION
or https://s3.region.amazonaws.com/amazoncloudwatch-agent-Region/info/latest/CWAGENT_VERSION

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux and Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://s3.amazonaws.com/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
		agent-<i>region</i>/suse/amd64/latest/amazon-cloudwatch-agent.rpm	agent-<i>region</i>/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://s3.amazonaws.com/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://s3.amazonaws.com/amazoncloudwatch-agent/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://s3.amazonaws.com/amazoncloudwatch-agent/darwin/amd64/latest/amazon-cloudwatch-agent.pkg <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/darwin/amd64/latest/amazon-cloudwatch-agent.pkg">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://s3.amazonaws.com/amazoncloudwatch-agent/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig <a href="https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig

Architecture	Platform	Download link	Signature file link
x86-64	Windows	https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
ARM64	SUSE	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

To use the command line to download and install the CloudWatch agent package

1. Download the CloudWatch agent.

On a Linux server, enter the following. For *download-link*, use the appropriate download link from the previous table.

```
wget download-link
```

On a server running Windows Server, download the following file:

```
https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi
```

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see [Verifying the signature of the CloudWatch agent package \(p. 515\)](#).
3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

If you downloaded an MSI package on a server running Windows Server, change to the directory containing the package and enter the following:

```
msiexec /i amazon-cloudwatch-agent.msi
```

This command also works from within PowerShell. For more information about MSI command options, see [Command-Line Options](#) in the Microsoft Windows documentation.

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#).

Create IAM roles and users for use with CloudWatch agent

Access to AWS resources requires permissions. You create an IAM role, an IAM user, or both to grant permissions that the CloudWatch agent needs to write metrics to CloudWatch. If you're going to use the agent on Amazon EC2 instances, you must create an IAM role. If you're going to use the agent on on-premises servers, you must create an IAM user.

Note

We recently modified the following procedures by using new `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies created by Amazon, instead of requiring customers to create these policies themselves. For writing files to and downloading files from the Parameter Store, the policies created by Amazon support only files with names that start with `AmazonCloudWatch-`. If you have a CloudWatch agent configuration file with a file name that doesn't start with `AmazonCloudWatch-`, these policies can't be used to write the file to Parameter Store or download it from Parameter Store.

If you're going to run the CloudWatch agent on Amazon EC2 instances, use the following steps to create the necessary IAM role. This role provides permissions for reading information from the instance and writing it to CloudWatch.

To create the IAM role necessary to run the CloudWatch agent on EC2 instances

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
 2. In the navigation pane on the left, choose **Roles** and then **Create role**.
 3. Make sure that **AWS service** is selected under **Trusted entity type**.
 4. For **Use case**, choose **EC2** under **Common use cases**,
 5. Choose **Next**.
 6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
 7. Choose **Next**.
 8. In **Role name**, enter a name for the role, such as `CloudWatchAgentServerRole`. Optionally give it a description. Then choose **Create role**.
- The role is now created.
9. (Optional) If the agent is going to send logs to CloudWatch Logs and you want the agent to be able to set retention policies for these log groups, you need to add the `logs:PutRetentionPolicy` permission to the role. For more information, see [Allowing the CloudWatch agent to set log retention policy \(p. 490\)](#).

If you're going to run the CloudWatch agent on on-premises servers, use the following steps to create the necessary IAM user.

To create the IAM user necessary for the CloudWatch agent to run on on-premises servers

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose **Users** and then **Add users**.
3. Enter the user name for the new user.
4. Select **Access key - Programmatic access** and choose **Next: Permissions**.
5. Choose **Attach existing policies directly**.
6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. Choose **Next: Tags**.

8. Optionally create tags for the new IAM user, and then choose **Next:Review**.
9. Confirm that the correct policy is listed, and choose **Create user**.
10. Next to the name of the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Allowing the CloudWatch agent to set log retention policy

You can configure the CloudWatch agent to set the retention policy for log groups that it sends log events to. If you do this, you must grant the `logs:PutRetentionPolicy` to the IAM role or user that the agent uses. The agent uses an IAM role to run on Amazon EC2 instances, and uses an IAM user for on-premises servers.

To grant the CloudWatch agent's IAM role permission to set log retention policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. In the search box, Type the beginning of the name of the CloudWatch agent's IAM role. You chose this name when you created the role. It might be named `CloudWatchAgentServerRole`.

When you see the role, choose the name of the role.

4. In the **Permissions** tab, choose **Add permissions, Create inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

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

6. Choose **Review policy**.
7. For **Name**, enter `CloudWatchAgentPutLogsRetention` or something similar, and choose **Create policy**.

To grant the CloudWatch agent's IAM user permission to set log retention policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Users**.
3. In the search box, Type the beginning of the name of the CloudWatch agent's IAM user. You chose this name when you created the user.

When you see the user, choose the name of the user.

4. In the **Permissions** tab, choose **Add inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

```
{
```

```

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

```

6. Choose **Review policy**.
7. For **Name**, enter **CloudWatchAgentPutLogsRetention** or something similar, and choose **Create policy**.

Installing and running the CloudWatch agent on your servers

After you have created the agent configuration file that you want and created an IAM role or IAM user, use the following steps to install and run the agent on your servers, using that configuration. First, attach an IAM role or IAM user to the server that will run the agent. Then, on that server, download the agent package and start it using the agent configuration you created.

Download the CloudWatch agent package using an S3 download link

You need to install the agent on each server where you will run the agent.

Amazon Linux 2

The CloudWatch agent is available as a package in Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances \(p. 499\)](#).

```
sudo yum install amazon-cloudwatch-agent
```

All operating systems

On all supported operating systems, you can download and install the CloudWatch agent using the command line with an Amazon S3 download link as described in the following steps.

For each download link, there is a general link as well as links for each Region. For example, for Amazon Linux and Amazon Linux 2 and the x86-64 architecture, three of the valid download links are:

- https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://s3.us-east-1.amazonaws.com/amazoncloudwatch-agent-us-east-1/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://s3.eu-central-1.amazonaws.com/amazoncloudwatch-agent-eu-central-1/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux and Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
		<a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	<a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://s3.amazonaws.com/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/centos/amd64/latest/amazon-cloudwatch-agent.rpm">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/redhat/amd64/latest/amazon-cloudwatch-agent.rpm">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/suse/amd64/latest/amazon-cloudwatch-agent.rpm">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://s3.amazonaws.com/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig <a href="https://s3.region.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>region</i>/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
x86-64	Ubuntu	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://s3.amazonaws.com/amazoncloudwatch-agent/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://s3.amazonaws.com/amazoncloudwatch-agent/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://s3.amazonaws.com/amazoncloudwatch-agent/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig

Architecture	Platform	Download link	Signature file link
ARM64	Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	SUSE	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

To use the command line to install the CloudWatch agent on an Amazon EC2 instance

1. Download the CloudWatch agent. For a Linux server, enter the following. For *download-link*, use the appropriate download link from the previous table.

```
wget download-link
```

For a server running Windows Server, download the following file:

```
https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi
```

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see [Verifying the signature of the CloudWatch agent package \(p. 515\)](#).
3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

If you downloaded an MSI package on a server running Windows Server, change to the directory containing the package and enter the following:

```
msiexec /i amazon-cloudwatch-agent.msi
```

This command also works from within PowerShell. For more information about MSI command options, see [Command-Line Options](#) in the Microsoft Windows documentation.

(Installing on an EC2 instance) Attaching an IAM role

To enable the CloudWatch agent to send data from the instance, you must attach an IAM role to the instance. The role to attach is **CloudWatchAgentServerRole**.

For more information on attaching an IAM role to an instance, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide for Windows Instances*.

(Installing on an on-premises server) Specify IAM credentials and AWS Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier. For more information about creating this user, see [Create IAM roles and users for use with CloudWatch agent \(p. 489\)](#).

You also must specify the AWS Region to send the metrics to, using the `region` field in the `[AmazonCloudWatchAgent]` section of the AWS config file, as in the following example.

```
[profile AmazonCloudWatchAgent]
region = us-west-1
```

The following is an example of using the `aws configure` command to create a named profile for the CloudWatch agent. This example assumes that you are using the default profile name of `AmazonCloudWatchAgent`.

To create the **AmazonCloudWatchAgent** profile for the CloudWatch agent

1. If you haven't already done so, install the AWS Command Line Interface on the server. For more information, see [Installing the AWS CLI](#).
2. On Linux servers, enter the following command and follow the prompts:

```
sudo aws configure --profile AmazonCloudWatchAgent
```

On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

```
aws configure --profile AmazonCloudWatchAgent
```

Verify internet access

Your Amazon EC2 instances must have outbound internet access to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must add the CloudWatch endpoints for the appropriate Regions to the allow list. These endpoints are listed in [Amazon CloudWatch](#) in the *Amazon Web Services General Reference*.
- If you're using the agent to collect logs, you must add the CloudWatch Logs endpoints for the appropriate Regions to the allow list. These endpoints are listed in [Amazon CloudWatch Logs](#) in the *Amazon Web Services General Reference*.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must add the Systems Manager endpoints for the appropriate Regions to the allow list. These endpoints are listed in [AWS Systems Manager](#) in the *Amazon Web Services General Reference*.

(Optional) Modify the common configuration for proxy or Region information

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows.

```
# This common-config is used to configure items used for both ssm and cloudwatch access

## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##     Instance role is used for EC2 case by default.
##     AmazonCloudWatchAgent profile is used for the on-premises case by default.
# [credentials]
#   shared_credential_profile = "{profile_name}"
#   shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
```

```
## Note: system-wide environment-variable is not accessible when using ssm run-command.  
## Absent in both here and environment-variable means no proxy will be used.  
# [proxy]  
#   http_proxy = "{http_url}"  
#   https_proxy = "{https_url}"  
#   no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually or by using the [RunShellScript Run Command](#) in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see [\(Installing on an on-premises server\) Specify IAM credentials and AWS Region \(p. 495\)](#).

On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\\\Users\\\\Administrator\\\\.aws` on Windows Server.

The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the \ characters.

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\\\Documents and Settings\\\\username\\\\.aws\\\\credentials"
```

If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Start the CloudWatch agent using the command line

Follow these steps to use the command line to start the CloudWatch agent on a server.

To use the command line to start the CloudWatch agent on a server

1. Copy the agent configuration file that you want to use to the server where you're going to run the agent. Note the pathname where you copy it to.
2. In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

Enter one of the following commands. Replace `configuration-file-path` with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

On an EC2 instance running Linux, enter the following command.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

On an on-premises server running Linux, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
```

On an EC2 instance running Windows Server, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m ec2 -s -c file:configuration-file-path
```

On an on-premises server running Windows Server, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m onPremise -s -c file:configuration-file-path
```

Installing the CloudWatch agent using AWS Systems Manager

Use the following topics to install and run the CloudWatch agent using AWS Systems Manager.

Topics

- [Create IAM roles and users for use with the CloudWatch agent \(p. 498\)](#)
- [Download and configure the CloudWatch agent \(p. 501\)](#)
- [Installing the CloudWatch agent on EC2 instances using your agent configuration \(p. 503\)](#)
- [Installing the CloudWatch agent on on-premises servers \(p. 507\)](#)

Create IAM roles and users for use with the CloudWatch agent

Access to AWS resources requires permissions. You can create IAM roles and users that include the permissions that you need for the CloudWatch agent to write metrics to CloudWatch and for the CloudWatch agent to communicate with Amazon EC2 and AWS Systems Manager. You use IAM roles on Amazon EC2 instances, and you use IAM users with on-premises servers.

One role or user enables CloudWatch agent to be installed on a server and send metrics to CloudWatch. The other role or user is needed to store your CloudWatch agent configuration in Systems Manager Parameter Store. Parameter Store enables multiple servers to use one CloudWatch agent configuration.

The ability to write to Parameter Store is a broad and powerful permission. You should use it only when you need it, and it shouldn't be attached to multiple instances in your deployment. If you store your CloudWatch agent configuration in Parameter Store, we recommend the following:

- Set up one instance where you perform this configuration.
- Use the IAM role with permissions to write to Parameter Store only on this instance.
- Use the IAM role with permissions to write to Parameter Store only while you are working with and saving the CloudWatch agent configuration file.

Note

We recently modified the following procedures by using new `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies created by Amazon, instead of requiring customers to create these policies themselves. To use these policies to write the agent configuration file to Parameter Store and then download it from Parameter Store, your agent configuration file must have a name that starts with `AmazonCloudWatch-`. If you have a CloudWatch agent configuration file with a file name that doesn't start with `AmazonCloudWatch-`, these policies can't be used to write the file to Parameter Store or to download the file from Parameter Store.

Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances

The first procedure creates the IAM role that you must attach to each Amazon EC2 instance that runs the CloudWatch agent. This role provides permissions for reading information from the instance and writing it to CloudWatch.

The second procedure creates the IAM role that you must attach to the Amazon EC2 instance being used to create the CloudWatch agent configuration file. This step is necessary if you're going to store this file in Systems Manager Parameter Store so that other servers can use it. This role provides permissions for writing to Parameter Store, in addition to the permissions for reading information from the instance and writing it to CloudWatch. This role includes permissions sufficient to run the CloudWatch agent as well as to write to Parameter Store.

Note

Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of details available with the CloudWatch Agent predefined metric sets.

To create the IAM role necessary for each server to run the CloudWatch agent

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Immediately under **Common use cases**, choose **EC2**, and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
6. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use Systems Manager service core functionality. If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.
7. Choose **Next: Tags**.
8. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for your new role, such as `CloudWatchAgentServerRole` or another name that you prefer.
10. (Optional) For **Role description**, enter a description.
11. Confirm that **CloudWatchAgentServerPolicy** and optionally **AmazonSSMManagedInstanceCore** appear next to **Policies**.
12. Choose **Create role**.

The role is now created.

The following procedure creates the IAM role that can also write to Parameter Store. You can use this role to store the agent configuration file in Parameter Store so that other servers can retrieve it.

The permissions for writing to Parameter Store provide broad access. This role shouldn't be attached to all your servers, and only administrators should use it. After you create the agent configuration file and copy it to Parameter Store, you should detach this role from the instance and use `CloudWatchAgentServerRole` instead.

To create the IAM role for an administrator to write to Parameter Store

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Immediately under **Choose the service that will use this role**, choose **EC2**, and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **CloudWatchAgentAdminPolicy**. If necessary, use the search box to find the policy.
6. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use Systems Manager service core functionality. If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.
7. Choose **Next: Tags**.
8. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for your new role, such as **CloudWatchAgentAdminRole** or another name that you prefer.
10. (Optional) For **Role description**, enter a description.
11. Confirm that **CloudWatchAgentAdminPolicy** and optionally **AmazonSSMManagedInstanceCore** appear next to **Policies**.
12. Choose **Create role**.

The role is now created.

Create IAM users to use with the CloudWatch agent on on-premises servers

The first procedure creates the IAM user that you need to run the CloudWatch agent. This user provides permissions to send data to CloudWatch.

The second procedure creates the IAM user that you can use when creating the CloudWatch agent configuration file. Use this procedure to store this file in Systems Manager Parameter Store so that other servers can use it. This user provides permissions to write to Parameter Store, in addition to the permissions to write data to CloudWatch.

Note

Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of details available with the CloudWatch Agent predefined metric sets.

To create the IAM user necessary for the CloudWatch agent to write data to CloudWatch

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Enter the user name for the new user.

4. For **Access type**, select **Programmatic access**, and then choose **Next: Permissions**.
5. For **Set permissions**, choose **Attach existing policies directly**.
6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use Systems Manager service core functionality. (If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.)
8. Choose **Next: Tags**.
9. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
10. Confirm that the correct policies are listed, and then choose **Create user**.
11. In the row for the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

The following procedure creates the IAM user that can also write to Parameter Store. If you're going to store the agent configuration file in Parameter Store so that other servers can use it, you need to use this IAM user. This IAM user provides permissions for writing to Parameter Store. This user also provides the permissions for reading information from the instance and writing it to CloudWatch. The permissions for writing to Systems Manager Parameter Store provide broad access. This IAM user shouldn't be attached to all your servers, and only administrators should use it. You should use this IAM user only when you are storing the agent configuration file in Parameter Store.

To create the IAM user necessary to store the configuration file in Parameter Store and send information to CloudWatch

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Enter the user name for the new user.
4. For **Access type**, select **Programmatic access**, and then choose **Next: Permissions**.
5. For **Set permissions**, choose **Attach existing policies directly**.
6. In the list of policies, select the check box next to **CloudWatchAgentAdminPolicy**. If necessary, use the search box to find the policy.
7. To use Systems Manager to install or configure the CloudWatch agent, select the check box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use Systems Manager service core functionality. (If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.)
8. Choose **Next: Tags**.
9. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
10. Confirm that the correct policies are listed, and then choose **Create user**.
11. In the row for the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Download and configure the CloudWatch agent

This section explains how to use Systems Manager to download the agent and then how to create your agent configuration file. Before you can use Systems Manager to download the agent, you must make sure that the instance is configured correctly for Systems Manager.

Installing or updating SSM Agent

On an Amazon EC2 instance, the CloudWatch agent requires that the instance is running version 2.2.93.0 or later. Before you install the CloudWatch agent, update or install SSM Agent on the instance if you haven't already done so.

For information about installing or updating SSM Agent on an instance running Linux, see [Installing and Configuring SSM Agent on Linux Instances](#) in the *AWS Systems Manager User Guide*.

For information about installing or updating the SSM Agent, see [Working with SSM Agent](#) in the *AWS Systems Manager User Guide*.

(Optional) Verify Systems Manager prerequisites

Before you use Systems Manager Run Command to install and configure the CloudWatch agent, verify that your instances meet the minimum Systems Manager requirements. For more information, see [Systems Manager Prerequisites](#) in the *AWS Systems Manager User Guide*.

Verify internet access

Your Amazon EC2 instances must have outbound internet access to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must whitelist the CloudWatch endpoints for the appropriate Regions. These endpoints are listed in [Amazon CloudWatch](#) in the *Amazon Web Services General Reference*.
- If you're using the agent to collect logs, you must whitelist the CloudWatch Logs endpoints for the appropriate Regions. These endpoints are listed in [Amazon CloudWatch Logs](#) in the *Amazon Web Services General Reference*.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must whitelist the Systems Manager endpoints for the appropriate Regions. These endpoints are listed in [AWS Systems Manager](#) in the *Amazon Web Services General Reference*.

Use the following steps to download the CloudWatch agent package using Systems Manager.

To download the CloudWatch agent using Systems Manager

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance to install the CloudWatch agent on. If you don't see a specific instance, it might not be configured as a managed instance for use with Systems Manager. For more information, see [Setting Up AWS Systems Manager for Hybrid Environments](#) in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** field, enter **AmazonCloudWatchAgent**.

8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers.

If you're going to save your agent configuration file in the Systems Manager Parameter Store, you must use an EC2 instance to save to the Parameter Store. Additionally, you must first attach to that instance the `CloudWatchAgentAdminRole` IAM role. For more information about attaching roles, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide for Windows Instances*.

For more information about creating the CloudWatch agent configuration file, see [Create the CloudWatch agent configuration file \(p. 522\)](#).

Installing the CloudWatch agent on EC2 instances using your agent configuration

After you have a CloudWatch agent configuration saved in Parameter Store, you can use it when you install the agent on other servers.

Topics

- [Attach an IAM role to the instance \(p. 503\)](#)
- [Download the CloudWatch agent package on an Amazon EC2 instance \(p. 503\)](#)
- [\(Optional\) Modify the common configuration and named profile for CloudWatch agent \(p. 505\)](#)
- [Start the CloudWatch agent \(p. 506\)](#)

Attach an IAM role to the instance

You must attach the `CloudWatchAgentServerRole` IAM role to the EC2 instance to be able to run the CloudWatch agent on the instance. This role enables the CloudWatch agent to perform actions on the instance.

For more information, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide for Windows Instances*.

Download the CloudWatch agent package on an Amazon EC2 instance

You need to install the agent on each server where you will run the agent. The CloudWatch agent is available as a package in Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the `CloudWatchAgentServerPolicy` attached. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances \(p. 499\)](#).

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems, you can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link. For information about using an Amazon S3 download link, see [Download the CloudWatch agent package \(p. 484\)](#).

Note

When you install or update the CloudWatch agent, only the **Uninstall and reinstall** option is supported. You can't use the **In-place update** option.

[Download the CloudWatch agent on an Amazon EC2 instance Using Systems Manager](#)

Before you can use Systems Manager to install the CloudWatch agent, you must make sure that the instance is configured correctly for Systems Manager.

[Installing or updating SSM Agent](#)

On an Amazon EC2 instance, the CloudWatch agent requires that the instance is running version 2.2.93.0 or later. Before you install the CloudWatch agent, update or install SSM Agent on the instance if you haven't already done so.

For information about installing or updating SSM Agent on an instance running Linux, see [Installing and Configuring the SSM Agent on Linux Instances](#) in the *AWS Systems Manager User Guide*.

For information about installing or updating SSM Agent on an instance running Windows Server, see [Installing and Configuring SSM Agent on Windows Instances](#) in the *AWS Systems Manager User Guide*.

[\(Optional\) Verify Systems Manager prerequisites](#)

Before you use Systems Manager Run Command to install and configure the CloudWatch agent, verify that your instances meet the minimum Systems Manager requirements. For more information, see [Setting Up AWS Systems Manager](#) in the *AWS Systems Manager User Guide*.

[Verify internet access](#)

Your Amazon EC2 instances must have outbound internet access in order to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

[Download the CloudWatch agent package](#)

Systems Manager Run Command enables you to manage the configuration of your instances. You specify a Systems Manager document, specify parameters, and execute the command on one or more instances. SSM Agent on the instance processes the command and configures the instance as specified.

[To download the CloudWatch agent using Run Command](#)

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance on which to install the CloudWatch agent. If you do not see a specific instance, it might not be configured for Run Command. For more information, see [Setting Up AWS Systems Manager for Hybrid Environments](#) in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter **AmazonCloudWatchAgent**.
8. Keep **Version** set to **latest** to install the latest version of the agent.

9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

(Optional) Modify the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can use this file to optionally specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access

## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##           Instance role is used for EC2 case by default.
##           AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#   shared_credential_profile = "{profile_name}"
#   shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#   http_proxy = "{http_url}"
#   https_proxy = "{https_url}"
#   no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the `#` from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used.

On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

If you specify a `shared_credential_profile`, you must also remove the `#` from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the `\` characters.

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
```

If you specify a `shared_credential_file`, you must also remove the # from the beginning of the [credentials] line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Start the CloudWatch agent

You can start the agent using Systems Manager Run Command or the command line.

Start the CloudWatch agent using Systems Manager Run Command

Follow these steps to start the agent using Systems Manager Run Command.

To start the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.
-or-
If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **configure**.
7. In the **Optional Configuration Source** list, choose **ssm**.
8. In the **Optional Configuration Location** box, enter the name of the agent configuration file that you created and saved to Systems Manager Parameter Store, as explained in [Create the CloudWatch agent configuration file \(p. 522\)](#).
9. In the **Optional Restart** list, choose **yes** to start the agent after you have finished these steps.
10. Choose **Run**.
11. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully started.

Start the CloudWatch agent on an Amazon EC2 instance using the command line

Follow these steps to use the command line to install the CloudWatch agent on an Amazon EC2 instance.

To use the command line to start the CloudWatch agent on an Amazon EC2 instance

- In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

Linux and macOS: If you saved the configuration file in the Systems Manager Parameter Store, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c ssm:configuration-parameter-store-name
```

Linux and macOS: If you saved the configuration file on the local computer, enter the following command. Replace **configuration-file-path** with the path to the agent configuration file. This file is called config.json if you created it with the wizard, and might be called amazon-cloudwatch-agent.json if you created it manually.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

Windows Server: If you saved the agent configuration file in Systems Manager Parameter Store, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m ec2 -s -c ssm:configuration-parameter-store-name
```

Windows Server: If you saved the agent configuration file on the local computer, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent\config.json"
```

Installing the CloudWatch agent on on-premises servers

If you have downloaded the CloudWatch agent on one computer and created the agent configuration file you want, you can use that configuration file to install the agent on other on-premises servers.

Download the CloudWatch agent on an on-premises server

You can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link. For information about using an Amazon S3 download link, see [Download the CloudWatch agent package \(p. 484\)](#).

Download using Systems Manager

To use Systems Manager Run Command, you must register your on-premises server with Amazon EC2 Systems Manager. For more information, see [Setting Up Systems Manager in Hybrid Environments](#) in the [AWS Systems Manager User Guide](#).

If you have already registered your server, update SSM Agent to the latest version.

For information about updating SSM Agent on a server running Linux, see [Install SSM Agent for a Hybrid Environment \(Linux\)](#) in the [AWS Systems Manager User Guide](#).

For information about updating the SSM Agent on a server running Windows Server, see [Install SSM Agent for a Hybrid Environment \(Windows\)](#) in the [AWS Systems Manager User Guide](#).

To use the SSM Agent to download the CloudWatch agent package on an on-premises server

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

- If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, select the server to install the CloudWatch agent on. If you don't see a specific server, it might not be configured for Run Command. For more information, see [Setting Up AWS Systems Manager for Hybrid Environments](#) in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter **AmazonCloudWatchAgent**.
8. Keep **Version** blank to install the latest version of the agent.
9. Choose **Run**.

The agent package is downloaded, and the next steps are to configure and start it.

(Installing on an on-premises server) Specify IAM credentials and AWS Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier. For more information about creating this user, see [Create IAM roles and users for use with the CloudWatch agent \(p. 498\)](#).

You also must specify the AWS Region to send the metrics to, using the `region` field.

Following is an example of this file.

```
[AmazonCloudWatchAgent]
aws_access_key_id=my_access_key
aws_secret_access_key=my_secret_key
region = us-west-1
```

For `my_access_key` and `my_secret_key`, use the keys from the IAM user that doesn't have the permissions to write to Systems Manager Parameter Store. For more information about the IAM users needed for CloudWatch agent, see [Create IAM users to use with the CloudWatch agent on on-premises servers \(p. 500\)](#).

If you name this profile `AmazonCloudWatchAgent`, you don't need to do anything more. Optionally, you can give it a different name and specify that name as the value for `shared_credential_profile` in the `common-config.toml` file, which is explained in the following section.

Following is an example of using the **aws configure** command to create a named profile for the CloudWatch agent. This example assumes that you're using the default profile name of `AmazonCloudWatchAgent`.

To create the `AmazonCloudWatchAgent` profile for the CloudWatch agent

1. If you haven't already done so, install the AWS Command Line Interface on the server. For more information, see [Installing the AWS CLI](#).
2. On Linux servers, enter the following command and follow the prompts:

```
sudo aws configure --profile AmazonCloudWatchAgent
```

On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

```
aws configure --profile AmazonCloudWatchAgent
```

(Optional) Modifying the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access

## Configuration for shared credential.
## Default credential strategy will be used if it is absent here.
##           Instance role is used for EC2 case by default.
##           AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#   shared_credential_profile = "{profile_name}"
#   shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#   http_proxy = "{http_url}"
#   https_proxy = "{https_url}"
#   no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the `#` from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see [\(Installing on an on-premises server\) Specify IAM credentials and AWS Region \(p. 508\)](#).

On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

If you specify a `shared_credential_profile`, you must also remove the `#` from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the `\` characters.

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
```

If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Starting the CloudWatch agent

You can start the CloudWatch agent using either Systems Manager Run Command or the command line.

To use SSM Agent to start the CloudWatch agent on an on-premises server

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, select the instance where you installed the agent.
6. In the **Action** list, choose **configure**.
7. In the **Mode** list, choose **onPremise**.
8. In the **Optional Configuration Location** box, enter the name of the agent configuration file that you created with the wizard and stored in the Parameter Store.
9. Choose **Run**.

The agent starts with the configuration you specified in the configuration file.

To use the command line to start the CloudWatch agent on an on-premises server

- In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

Linux: If you saved the configuration file in the Systems Manager Parameter Store, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Linux: If you saved the configuration file on the local computer, enter the following command. Replace `configuration-file-path` with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
```

Windows Server: If you saved the agent configuration file in Systems Manager Parameter Store, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a  
fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Windows Server: If you saved the agent configuration file on the local computer, enter the following from the PowerShell console. Replace **configuration-file-path** with the path to the agent configuration file. This file is called config.json if you created it with the wizard, and might be called amazon-cloudwatch-agent.json if you created it manually.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a  
fetch-config -m onPremise -s -c file:configuration-file-path
```

Installing the CloudWatch agent on new instances using AWS CloudFormation

Amazon has uploaded several AWS CloudFormation templates to GitHub to help you install and update the CloudWatch agent on new Amazon EC2 instances. For more information about using AWS CloudFormation, see [What is AWS CloudFormation?](#)

The template location is [Deploy the Amazon CloudWatch agent to EC2 instances using AWS CloudFormation](#). This location includes both inline and ssm directories. Each of these directories contains templates for both Linux and Windows instances.

- The templates in the inline directory have the CloudWatch agent configuration embedded into the AWS CloudFormation template. By default, the Linux templates collect the metrics mem_used_percent and swap_used_percent, and the Windows templates collect Memory % Committed Bytes In Use and Paging File % Usage.

To modify these templates to collect different metrics, modify the following section of the template. The following example is from the template for Linux servers. Follow the format and syntax of the agent configuration file to make these changes. For more information, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

```
{  
    "metrics":{  
        "append_dimensions":{  
            "AutoScalingGroupName": "${!aws:AutoScalingGroupName}",  
            "ImageId": "${!aws:ImageId}",  
            "InstanceId": "${!aws:InstanceId}",  
            "InstanceType": "${!aws:InstanceType}"  
        },  
        "metrics_collected":{  
            "mem":{  
                "measurement": [  
                    "mem_used_percent"  
                ]  
            },  
            "swap":{  
                "measurement": [  
                    "swap_used_percent"  
                ]  
            }  
        }  
    }  
}
```

```
    }  
}
```

Note

In the inline templates, all placeholder variables must have an exclamation mark (!) before them as an escape character. You can see this in the example template. If you add other placeholder variables, be sure to add an exclamation mark before the name.

- The templates in the `ssm` directory load an agent configuration file from Parameter Store. To use these templates, you must first create a configuration file and upload it to Parameter Store. You then provide the Parameter Store name of the file in the template. You can create the configuration file manually or by using the wizard. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#).

You can use both types of templates for installing the CloudWatch agent and for updating the agent configuration.

Tutorial: Install and configure the CloudWatch agent using an AWS CloudFormation inline template

This tutorial walks you through using AWS CloudFormation to install the CloudWatch agent on a new Amazon EC2 instance. This tutorial installs on a new instance running Amazon Linux 2 using the inline templates, which don't require the use of the JSON configuration file or Parameter Store. The inline template includes the agent configuration in the template. In this tutorial, you use the default agent configuration contained in the template.

After the procedure for installing the agent, the tutorial continues with how to update the agent.

To use AWS CloudFormation to install the CloudWatch agent on a new instance

- Download the template from GitHub. In this tutorial, download the inline template for Amazon Linux 2 as follows:

```
curl -O https://raw.githubusercontent.com/aws-labs/aws-cloudformation-templates/master/  
aws/solutions/AmazonCloudWatchAgent/inline/amazon_linux.template
```

- Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
- Choose **Create stack**.
- For **Choose a template**, select **Upload a template to Amazon S3**, choose the downloaded template, and choose **Next**.
- On the **Specify Details** page, fill out the following parameters and choose **Next**:
 - Stack name**: Choose a stack name for your AWS CloudFormation stack.
 - IAMRole**: Choose an IAM role that has permissions to write CloudWatch metrics and logs. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances \(p. 489\)](#).
 - InstanceAMI**: Choose an AMI that is valid in the Region where you're going to launch your stack.
 - InstanceType**: Choose a valid instance type.
 - KeyName**: To enable SSH access to the new instance, choose an existing Amazon EC2 key pair. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.
 - SSHLocation**: Specifies the IP address range that can be used to connect to the instance using SSH. The default allows access from any IP address.

6. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
7. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

If you refresh the console, you see that the new stack has the `CREATE_IN_PROGRESS` status.
8. When the instance is created, you can see it in the Amazon EC2 console. Optionally, you can connect to the host and check the progress.

Use the following command to confirm that the agent is installed:

```
rpm -qa amazon-cloudwatch-agent
```

Use the following command to confirm that the agent is running:

```
ps aux | grep amazon-cloudwatch-agent
```

The next procedure demonstrates using AWS CloudFormation to update the CloudWatch agent using an inline template. The default inline template collects the `mem_used_percent` metric. In this tutorial, you change the agent configuration to stop collecting that metric.

To use AWS CloudFormation to update the CloudWatch agent

1. In the template that you downloaded in the previous procedure, remove the following lines and then save the template:

```
"mem": {  
    "measurement": [  
        "mem_used_percent"  
    ]  
},
```

2. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
3. On the AWS CloudFormation dashboard, select the stack that you created and choose **Update Stack**.
4. For **Select Template**, select **Upload a template to Amazon S3**, choose the template that you modified, and choose **Next**.
5. On the **Options** page, choose **Next** and then **Next**.
6. On the **Review** page, review your information and choose **Update**.

After some time, you see `UPDATE_COMPLETE`.

Tutorial: Install the CloudWatch agent using AWS CloudFormation and Parameter Store

This tutorial walks you through using AWS CloudFormation to install the CloudWatch agent on a new Amazon EC2 instance. This tutorial installs on a new instance running Amazon Linux 2 using an agent configuration file that you create and save in Parameter Store.

After the procedure for installing the agent, the tutorial continues with how to update the agent.

To use AWS CloudFormation to install the CloudWatch agent on a new instance using a configuration from Parameter Store

1. If you haven't done so already, download the CloudWatch agent package to one of your computers so that you can create the agent configuration file. For more information and downloading the agent using Parameter Store, see [Download and configure the CloudWatch agent \(p. 501\)](#). For more information on downloading the package using the command line, see [Download and configure the CloudWatch agent using the command line \(p. 484\)](#).
2. Create the agent configuration file and save it in Parameter Store. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#).
3. Download the template from GitHub as follows:

```
curl -O https://raw.githubusercontent.com/aws-labs/aws-cloudformation-templates/master/aws/solutions/AmazonCloudWatchAgent/ssm/amazon_linux.template
```

4. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
5. Choose **Create stack**.
6. For **Choose a template**, select **Upload a template to Amazon S3**, choose the template that you downloaded, and choose **Next**.
7. On the **Specify Details** page, fill out the following parameters accordingly and choose **Next**:
 - **Stack name**: Choose a stack name for your AWS CloudFormation stack.
 - **IAMRole**: Choose an IAM role that has permissions to write CloudWatch metrics and logs. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances \(p. 499\)](#).
 - **InstanceAMI**: Choose an AMI that is valid in the Region where you're going to launch your stack.
 - **InstanceType**: Choose a valid instance type.
 - **KeyName**: To enable SSH access to the new instance, choose an existing Amazon EC2 key pair. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 Key Pairs](#) in the [Amazon EC2 User Guide for Linux Instances](#).
 - **SSHLlocation**: Specifies the IP address range that can be used to connect to the instance using SSH. The default allows access from any IP address.
 - **SSMKey**: Specifies the agent configuration file that you created and saved in Parameter Store.
8. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
9. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

If you refresh the console, you see that the new stack has the `CREATE_IN_PROGRESS` status.

10. When the instance is created, you can see it in the Amazon EC2 console. Optionally, you can connect to the host and check the progress.

Use the following command to confirm that the agent is installed:

```
rpm -qa amazon-cloudwatch-agent
```

Use the following command to confirm that the agent is running:

```
ps aux | grep amazon-cloudwatch-agent
```

The next procedure demonstrates using AWS CloudFormation to update the CloudWatch agent, using an agent configuration that you saved in Parameter Store.

To use AWS CloudFormation to update the CloudWatch agent using a configuration in Parameter Store

1. Change the agent configuration file stored in Parameter Store to the new configuration that you want.
2. In the AWS CloudFormation template that you downloaded in the [the section called “Tutorial: Install the CloudWatch agent using AWS CloudFormation and Parameter Store” \(p. 513\)](#) topic, change the version number. For example, you might change VERSION=1.0 to VERSION=2.0.
3. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
4. On the AWS CloudFormation dashboard, select the stack that you created and choose **Update Stack**.
5. For **Select Template**, select **Upload a template to Amazon S3**, select the template that you just modified, and choose **Next**.
6. On the **Options** page, choose **Next** and then **Next**.
7. On the **Review** page, review your information and choose **Update**.

After some time, you see **UPDATE_COMPLETE**.

Troubleshooting installation of the CloudWatch agent with AWS CloudFormation

This section helps you troubleshoot issues with installing and updating the CloudWatch agent using AWS CloudFormation.

Detecting when an update fails

If you use AWS CloudFormation to update your CloudWatch agent configuration, and use an invalid configuration, the agent stops sending any metrics to CloudWatch. A quick way to check whether an agent configuration update succeeded is to look at the `cfn-init-cmd.log` file. On a Linux server, the file is located at `/var/log/cfn-init-cmd.log`. On a Windows instance, the file is located at `C:\log\cfn-init-cmd.log`.

Metrics are missing

If you don't see metrics that you expect to see after installing or updating the agent, confirm that the agent is configured to collect that metric. To do this, check the `amazon-cloudwatch-agent.json` file to make sure that the metric is listed, and check that you are looking in the correct metric namespace. For more information, see [CloudWatch agent files and locations \(p. 599\)](#).

Verifying the signature of the CloudWatch agent package

GPG signature files are included for CloudWatch agent packages on Linux servers. You can use a public key to verify that the agent download file is original and unmodified.

For Windows Server, you can use the MSI to verify the signature.

For macOS computers, the signature is included in the agent download package.

To find the correct signature file, see the following table. For each architecture and operating system there is a general link as well as links for each Region. For example, for Amazon Linux and Amazon Linux 2 and the x86-64 architecture, three of the valid links are:

- https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
- https://s3.us-east-1.amazonaws.com/amazoncloudwatch-agent-us-east-1/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
- https://s3.eu-central-1.amazonaws.com/amazoncloudwatch-agent-eu-central-1/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux and Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://s3.amazonaws.com/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/amd64/	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/amd64/

Architecture	Platform	Download link	Signature file link
		latest/amazon-cloudwatch-agent.rpm	latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://s3.amazonaws.com/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://s3.amazonaws.com/amazoncloudwatch-agent/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://s3.amazonaws.com/amazoncloudwatch-agent/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://s3.amazonaws.com/amazoncloudwatch-agent/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig

Architecture	Platform	Download link	Signature file link
x86-64	Windows	https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://s3.amazonaws.com/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
ARM64	SUSE	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

To verify the CloudWatch agent package on a Linux server

1. Download the public key.

```
shell$ wget https://s3.amazonaws.com/amazoncloudwatch-agent/assets/amazon-cloudwatch-agent.gpg
```

2. Import the public key into your keyring.

```
shell$ gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

Make a note of the key value, as you need it in the next step. In the preceding example, the key value is 3B789C72.

3. Verify the fingerprint by running the following command, replacing **key-value** with the value from the preceding step:

```
shell$ gpg --fingerprint key-value
pub    2048R/3B789C72 2017-11-14
      Key fingerprint = 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid          Amazon CloudWatch Agent
```

The fingerprint string should be equal to the following:

9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

4. Download the package signature file using **wget**. To determine the correct signature file, see the preceding table.

```
wget Signature File Link
```

5. To verify the signature, run **gpg --verify**.

```
shell$ gpg --verify signature-filename agent-download-filename
gpg: Signature made Wed 29 Nov 2017 03:00:59 PM PST using RSA key ID 3B789C72
gpg: Good signature from "Amazon CloudWatch Agent"
```

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                 There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

To verify the CloudWatch agent package on a server running Windows Server

1. Download and install GnuPG for Windows from <https://gnupg.org/download/>. When installing, include the **Shell Extension (GpgEx)** option.

You can perform the remaining steps in Windows PowerShell.

2. Download the public key.

```
PS> wget https://s3.amazonaws.com/amazoncloudwatch-agent/assets/amazon-cloudwatch-
agent.gpg -OutFile amazon-cloudwatch-agent.gpg
```

3. Import the public key into your keyring.

```
PS> gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

Make a note of the key value because you need it in the next step. In the preceding example, the key value is `3B789C72`.

4. Verify the fingerprint by running the following command, replacing `key-value` with the value from the preceding step:

```
PS> gpg --fingerprint key-value
pub    rsa2048 2017-11-14 [SC]
      9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid          [ unknown] Amazon CloudWatch Agent
```

The fingerprint string should be equal to the following:

9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

5. Download the package signature file using `wget`. To determine the correct signature file, see [CloudWatch Agent Download Links \(p. 484\)](#).
6. To verify the signature, run `gpg --verify`.

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:                               using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
```

```
gpg: There is no indication that the signature belongs to the owner.  
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

To verify the CloudWatch agent package on a macOS computer

- There are two methods for signature verification on macOS.
 - Verify the fingerprint by running the following command.

```
pkgutil --check-signature amazon-cloudwatch-agent.pkg
```

You should see a result similar to the following.

```
Package "amazon-cloudwatch-agent.pkg":  
  Status: signed by a developer certificate issued by Apple for distribution  
  Signed with a trusted timestamp on: 2020-10-02 18:13:24 +0000  
  Certificate Chain:  
    1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)  
       Expires: 2024-10-18 22:31:30 +0000  
       SHA256 Fingerprint:  
         81 B4 6F AF 1C CA E1 E8 3C 6F FB 9E 52 5E 84 02 6E 7F 17 21 8E FB  
         0C 40 79 13 66 8D 9F 1F 10 1C  
-----  
    2. Developer ID Certification Authority  
       Expires: 2027-02-01 22:12:15 +0000  
       SHA256 Fingerprint:  
         7A FC 9D 01 A6 2F 03 A2 DE 96 37 93 6D 4A FE 68 09 0D 2D E1 8D 03  
         F2 9C 88 CF B0 B1 BA 63 58 7F  
-----  
    3. Apple Root CA  
       Expires: 2035-02-09 21:40:36 +0000  
       SHA256 Fingerprint:  
         B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C  
         68 C5 BE 91 B5 A1 10 01 F0 24
```

- Or, download and use the .sig file To use this method, follow these steps.
 - Install the GPG application to your macOS host by entering the following command.

```
brew install GnuPG
```

- Download the package signature file using curl. To determine the correct signature file, see [CloudWatch Agent Download Links \(p. 484\)](#).
- To verify the signature, run `gpg --verify`.

```
PS> gpg --verify sig-filename agent-download-filename  
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time  
gpg:                               using RSA key D58167303B789C72  
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]  
gpg: WARNING: This key is not certified with a trusted signature!  
gpg:                               There is no indication that the signature belongs to the owner.  
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Create the CloudWatch agent configuration file

Before running the CloudWatch agent on any servers, you must create a CloudWatch agent configuration file.

The agent configuration file is a JSON file that specifies the metrics and logs that the agent is to collect, including custom metrics. You can create it by using the wizard or by creating it yourself from scratch. You could also use the wizard to initially create the configuration file and then modify it manually. If you create or modify the file manually, the process is more complex, but you have more control over the metrics collected and can specify metrics not available through the wizard.

Any time you change the agent configuration file, you must then restart the agent to have the changes take effect. To restart the agent, follow the instructions in [Start the CloudWatch agent \(p. 506\)](#).

After you have created a configuration file, you can save it manually as a JSON file and then use this file when installing the agent on your servers. Alternatively, you can store it in Systems Manager Parameter Store if you're going to use Systems Manager when you install the agent on servers.

Contents

- [Create the CloudWatch agent configuration file with the wizard \(p. 522\)](#)
- [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#)

Create the CloudWatch agent configuration file with the wizard

The agent configuration file wizard, `amazon-cloudwatch-agent-config-wizard`, asks a series of questions, including the following:

- Are you installing the agent on an Amazon EC2 instance or an on-premises server?
- Is the server running Linux or Windows Server?
- Do you want the agent to also send log files to CloudWatch Logs? If so, do you have an existing CloudWatch Logs agent configuration file? If yes, the CloudWatch agent can use this file to determine the logs to collect from the server.
- If you're going to collect metrics from the server, do you want to monitor one of the default sets of metrics or customize the list of metrics that you collect?
- Do you want to collect custom metrics from your applications or services, using `StatsD` or `collectd`?
- Are you migrating from an existing SSM Agent?

The wizard can autodetect the credentials and AWS Region to use if you have the AWS credentials and configuration files in place before you start the wizard. For more information about these files, see [Configuration and Credential Files](#) in the [AWS Systems Manager User Guide](#).

In the AWS credentials file, the wizard checks for default credentials and also looks for an `AmazonCloudWatchAgent` section such as the following:

```
[AmazonCloudWatchAgent]
aws_access_key_id = my_access_key
aws_secret_access_key = my_secret_key
```

The wizard displays the default credentials, the credentials from the `AmazonCloudWatchAgent`, and an `Others` option. You can select which credentials to use. If you choose `Others`, you can input credentials.

For `my_access_key` and `my_secret_key`, use the keys from the IAM user that has the permissions to write to Systems Manager Parameter Store. For more information about the IAM users needed for the CloudWatch agent, see [Create IAM users to use with the CloudWatch agent on on-premises servers \(p. 500\)](#).

In the AWS configuration file, you can specify the Region that the agent sends metrics to if it's different than the `[default]` section. The default is to publish the metrics to the Region where the Amazon EC2 instance is located. If the metrics should be published to a different Region, specify the Region here. In the following example, the metrics are published to the `us-west-1` Region.

```
[AmazonCloudWatchAgent]
region = us-west-1
```

CloudWatch agent predefined metric sets

The wizard is configured with predefined sets of metrics, with different detail levels. These sets of metrics are shown in the following tables. For more information about these metrics, see [Metrics collected by the CloudWatch agent \(p. 574\)](#).

Note

Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of metric details that are described in these tables.

Amazon EC2 instances running Linux

Detail level	Metrics included
Basic	Mem: <code>mem_used_percent</code> Disk: <code>disk_used_percent</code> The disk metrics such as <code>disk_used_percent</code> have a dimension for <code>Partition</code> , which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.
Standard	CPU: <code>cpu_usage_idle</code> , <code>cpu_usage_iowait</code> , <code>cpu_usage_user</code> , <code>cpu_usage_system</code> Disk: <code>disk_used_percent</code> , <code>disk_inodes_free</code> Diskio: <code>diskio_io_time</code> Mem: <code>mem_used_percent</code> Swap: <code>swap_used_percent</code>
Advanced	CPU: <code>cpu_usage_idle</code> , <code>cpu_usage_iowait</code> , <code>cpu_usage_user</code> , <code>cpu_usage_system</code>

Detail level	Metrics included
	Disk: disk_used_percent, disk_inodes_free Diskio: diskio_io_time, diskio_write_bytes, diskio_read_bytes, diskio_writes, diskio_reads Mem: mem_used_percent Netstat: netstat_tcp_established, netstat_tcp_time_wait Swap: swap_used_percent

On-premises servers running Linux

Detail level	Metrics included
Basic	Disk: disk_used_percent Diskio: diskio_write_bytes, diskio_read_bytes, diskio_writes, diskio_reads Mem: mem_used_percent Net: net_bytes_sent, net_bytes_recv, net_packets_sent, net_packets_recv Swap: swap_used_percent
Standard	CPU: cpu_usage_idle, cpu_usage_iowait Disk: disk_used_percent, disk_inodes_free Diskio: diskio_io_time, diskio_write_bytes, diskio_read_bytes, diskio_writes, diskio_reads Mem: mem_used_percent Net: net_bytes_sent, net_bytes_recv, net_packets_sent, net_packets_recv Swap: swap_used_percent
Advanced	CPU: cpu_usage_guest, cpu_usage_idle, cpu_usage_iowait, cpu_usage_stolen, cpu_usage_user, cpu_usage_system Disk: disk_used_percent, disk_inodes_free Diskio: diskio_io_time, diskio_write_bytes, diskio_read_bytes, diskio_writes, diskio_reads Mem: mem_used_percent Net: net_bytes_sent, net_bytes_recv, net_packets_sent, net_packets_recv Netstat: netstat_tcp_established, netstat_tcp_time_wait Swap: swap_used_percent

Amazon EC2 instances running Windows Server

Detail level	Metrics included
Basic	Memory: Memory % Committed Bytes In Use LogicalDisk: LogicalDisk % Free Space
Standard	Memory: Memory % Committed Bytes In Use Paging: Paging File % Usage Processor: Processor % Idle Time, Processor % Interrupt Time, Processor % User Time PhysicalDisk: PhysicalDisk % Disk Time LogicalDisk: LogicalDisk % Free Space
Advanced	Memory: Memory % Committed Bytes In Use Paging: Paging File % Usage Processor: Processor % Idle Time, Processor % Interrupt Time, Processor % User Time LogicalDisk: LogicalDisk % Free Space PhysicalDisk: PhysicalDisk % Disk Time, PhysicalDisk Disk Write Bytes/sec, PhysicalDisk Disk Read Bytes/sec, PhysicalDisk Disk Writes/sec, PhysicalDisk Disk Reads/sec TCP: TCPv4 Connections Established, TCPv6 Connections Established

On-premises server running Windows Server

Detail level	Metrics included
Basic	Paging: Paging File % Usage Processor: Processor % Processor Time LogicalDisk: LogicalDisk % Free Space PhysicalDisk: PhysicalDisk Disk Write Bytes/sec, PhysicalDisk Disk Read Bytes/sec, PhysicalDisk Disk Writes/sec, PhysicalDisk Disk Reads/sec Memory: Memory % Committed Bytes In Use Network Interface: Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec, Network Interface Packets Sent/sec, Network Interface Packets Received/sec
Standard	Paging: Paging File % Usage Processor: Processor % Processor Time, Processor % Idle Time, Processor % Interrupt Time

Detail level	Metrics included
	<p>LogicalDisk: LogicalDisk % Free Space</p> <p>PhysicalDisk: PhysicalDisk % Disk Time, PhysicalDisk Disk Write Bytes/sec, PhysicalDisk Disk Read Bytes/sec, PhysicalDisk Disk Writes/sec, PhysicalDisk Disk Reads/sec</p> <p>Memory: Memory % Committed Bytes In Use</p> <p>Network Interface: Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec, Network Interface Packets Sent/sec, Network Interface Packets Received/sec</p>
Advanced	<p>Paging: Paging File % Usage</p> <p>Processor: Processor % Processor Time, Processor % Idle Time, Processor % Interrupt Time, Processor % User Time</p> <p>LogicalDisk: LogicalDisk % Free Space</p> <p>PhysicalDisk: PhysicalDisk % Disk Time, PhysicalDisk Disk Write Bytes/sec, PhysicalDisk Disk Read Bytes/sec, PhysicalDisk Disk Writes/sec, PhysicalDisk Disk Reads/sec</p> <p>Memory: Memory % Committed Bytes In Use</p> <p>Network Interface: Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec, Network Interface Packets Sent/sec, Network Interface Packets Received/sec</p> <p>TCP: TCPv4 Connections Established, TCPv6 Connections Established</p>

Run the CloudWatch agent configuration wizard

To create the CloudWatch agent configuration file

1. Start the CloudWatch agent configuration wizard by entering the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
```

On a server running Windows Server, enter the following:

```
cd "C:\Program Files\Amazon\AmazonCloudWatchAgent"
amazon-cloudwatch-agent-config-wizard.exe
```

2. Answer the questions to customize the configuration file for your server.
3. If you're storing the configuration file locally, the configuration file config.json is stored in /opt/aws/amazon-cloudwatch-agent/bin/ on Linux servers, and is stored in C:\Program Files\Amazon\AmazonCloudWatchAgent on Windows Server. You can then copy this file to other servers where you want to install the agent.

If you're going to use Systems Manager to install and configure the agent, be sure to answer **Yes** when prompted whether to store the file in Systems Manager Parameter Store. You can also choose to store the file in Parameter Store even if you aren't using the SSM Agent to install the CloudWatch agent. To be able to store the file in Parameter Store, you must use an IAM role with sufficient

permissions. For more information, see [Create IAM roles and users for use with the CloudWatch agent \(p. 498\)](#).

Manually create or edit the CloudWatch agent configuration file

The CloudWatch agent configuration file is a JSON file with three sections: `agent`, `metrics`, and `logs`.

- The `agent` section includes fields for the overall configuration of the agent. If you use the wizard, it doesn't create an `agent` section.
- The `metrics` section specifies the custom metrics for collection and publishing to CloudWatch. If you're using the agent only to collect logs, you can omit the `metrics` section from the file.
- The `logs` section specifies what log files are published to CloudWatch Logs. This can include events from the Windows Event Log if the server runs Windows Server.

The following sections explain the structure and fields of this JSON file. You can also view the schema definition for this configuration file. The schema definition is located at `installation-directory/doc/amazon-cloudwatch-agent-schema.json` on Linux servers, and at `installation-directory/amazon-cloudwatch-agent-schema.json` on servers running Windows Server.

If you create or edit the agent configuration file manually, you can give it any name. For simplicity in troubleshooting, we recommend that you name it `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` on a Linux server and `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json` on servers running Windows Server. After you have created the file, you can copy it to other servers where you want to install the agent.

CloudWatch agent configuration file: Agent section

The `agent` section can include the following fields. The wizard doesn't create an `agent` section. Instead, the wizard omits it and uses the default values for all fields in this section.

- `metrics_collection_interval` – Optional. Specifies how often all metrics specified in this configuration file are to be collected. You can override this value for specific types of metrics.

This value is specified in seconds. For example, specifying 10 sets metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

The default value is 60.

- `region` – Specifies the Region to use for the CloudWatch endpoint when an Amazon EC2 instance is being monitored. The metrics collected are sent to this Region, such as `us-west-1`. If you omit this field, the agent sends metrics to the Region where the Amazon EC2 instance is located.

If you are monitoring an on-premises server, this field isn't used, and the agent reads the Region from the `AmazonCloudWatchAgent` profile of the AWS configuration file.

- `credentials` – Specifies an IAM role to use when sending metrics and logs to a different AWS account. If specified, this field contains one parameter, `role_arn`.
 - `role_arn` – Specifies the Amazon Resource Name (ARN) of an IAM role to use for authentication when sending metrics and logs to a different AWS account. For more information, see [Sending metrics and logs to a different account \(p. 594\)](#).
- `debug` – Optional. Specifies running the CloudWatch agent with debug log messages. The default value is `false`.

- `aws_sdk_log_level` – Optional. Supported only in versions 1.247350.0 and later of the CloudWatch agent,

You can specify this field to have the agent perform logging for AWS SDK endpoints. The value for this field can include one or more of the following options. Separate multiple options with the `|` character.

- `LogDebug`
- `LogDebugWithSigning`
- `LogDebugWithHTTPBody`
- `LogDebugRequestRetries`
- `LogDebugWithEventStreamBody`
- `LogDebug`

For more information about these options, see [LogLevelType](#).

- `logfile` – Specifies the location where the CloudWatch agent writes log messages. If you specify an empty string, the log goes to stderr. If you don't specify this option, the default locations are the following:
 - Linux: `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log`
 - Windows Server: `c:\ProgramData\Amazon\CloudWatchAgent\Logs\amazon-cloudwatch-agent.log`

The CloudWatch agent automatically rotates the log file that it creates. A log file is rotated out when it reaches 100 MB in size. The agent keeps the rotated log files for up to seven days, and it keeps as many as five backup log files that have been rotated out. Backup log files have a timestamp appended to their filename. The timestamp shows the date and time that the file was rotated out: for example, `amazon-cloudwatch-agent-2018-06-08T21-01-50.247.log.gz`.

- `omit_hostname` – Optional. By default, the hostname is published as a dimension of metrics that are collected by the agent, unless you are using the `append_dimensions` field in the `metrics` section. Set `omit_hostname` to `true` to prevent the hostname from being published as a dimension even if you are not using `append_dimensions`. The default value is `false`.
- `run_as_user` – Optional. Specifies a user to use to run the CloudWatch agent. If you don't specify this parameter, the root user is used. This option is valid only on Linux servers.

If you specify this option, the user must exist before you start the CloudWatch agent. For more information, see [Running the CloudWatch agent as a different user \(p. 589\)](#).

- `user_agent` – Optional. Specifies the `user-agent` string that is used by the CloudWatch agent when it makes API calls to the CloudWatch backend. The default value is a string consisting of the agent version, the version of the Go programming language that was used to compile the agent, the runtime operating system and architecture, the build time, and the plugins enabled.

The following is an example of an agent section.

```
"agent": {  
    "metrics_collection_interval": 60,  
    "region": "us-west-1",  
    "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",  
    "debug": false,  
    "run_as_user": "cwagent"  
}
```

CloudWatch agent configuration file: Metrics section

On servers running either Linux or Windows Server, the `metrics` section includes the following fields:

- **namespace** – Optional. The namespace to use for the metrics collected by the agent. The default value is CWAgent. The maximum length is 255 characters. The following is an example:

```
{  
  "metrics": {  
    "namespace": "Development/Product1Metrics",  
    .....  
  },  
}
```

- **append_dimensions** – Optional. Adds Amazon EC2 metric dimensions to all metrics collected by the agent. This also causes the agent to not publish the hostname as a dimension.

The only supported key-value pairs for `append_dimensions` are shown in the following list. Any other key-value pairs are ignored.

- `"ImageID": "${aws:ImageId}"` sets the instance's AMI ID as the value of the `ImageID` dimension.
- `"InstanceId": "${aws:InstanceId}"` sets the instance's instance ID as the value of the `InstanceId` dimension.
- `"InstanceType": "${aws:InstanceType}"` sets the instance's instance type as the value of the `InstanceType` dimension.
- `"AutoScalingGroupName": "${aws:AutoScalingGroupName}"` sets the instance's Auto Scaling group name as the value of the `AutoScalingGroupName` dimension.

If you want to append dimensions to metrics with arbitrary key-value pairs, use the `append_dimensions` parameter in the field for that particular type of metric.

If you specify a value that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see [Amazon Elastic Compute Cloud \(Amazon EC2\)](#) in the [Amazon Web Services General Reference](#).

- **aggregation_dimensions** – Optional. Specifies the dimensions that collected metrics are to be aggregated on. For example, if you roll up metrics on the `AutoScalingGroupName` dimension, the metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

You can roll up metrics along single or multiple dimensions. For example, specifying `[["InstanceId"], ["InstanceType"], ["InstanceId", "InstanceType"]]` aggregates metrics for instance ID singly, instance type singly, and for the combination of the two dimensions.

You can also specify `[]` to roll up all metrics into one collection, disregarding all dimensions.

- **endpoint_override** – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends metrics. Specifying this and setting a private link enables you to send the metrics to an Amazon VPC endpoint. For more information, see [What Is Amazon VPC?](#).

The value of `endpoint_override` must be a string that is a URL.

For example, the following part of the metrics section of the configuration file sets the agent to use a VPC Endpoint when sending metrics.

```
{  
  "metrics": {  
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXX.monitoring.us-  
east-1.vpce.amazonaws.com",  
    .....  
  },  
}
```

- **metrics_collected** – Required. Specifies which metrics are to be collected, including custom metrics collected through StatsD or collectd. This section includes several subsections.

The contents of the `metrics_collected` section depend on whether this configuration file is for a server running Linux or Windows Server.

- `force_flush_interval` – Specifies in seconds the maximum amount of time that metrics remain in the memory buffer before being sent to the server. No matter the setting for this, if the size of the metrics in the buffer reaches 40 KB or 20 different metrics, the metrics are immediately sent to the server.

The default value is 60.

- `credentials` – Specifies an IAM role to use when sending metrics to a different account. If specified, this field contains one parameter, `role_arn`.
 - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending metrics to a different account. For more information, see [Sending metrics and logs to a different account \(p. 594\)](#). If specified here, this value overrides the `role_arn` specified in the `agent` section of the configuration file, if any.

Linux section

On servers running Linux, the `metrics_collected` section of the configuration file can also contain the following fields.

Many of these fields can include a `measurement` sections that lists the metrics you want to collect for that resource. These `measurement` sections can either specify the complete metric name such as `swap_used`, or just the part of the metric name that will be appended to the type of resource. For example, specifying `reads` in the `measurement` section of the `diskio` section causes the `diskio_reads` metric to be collected.

- `collectd` – Optional. Specifies that you want to retrieve custom metrics using the `collectd` protocol. You use `collectd` software to send the metrics to the CloudWatch agent. For more information about the configuration options available for `collectd`, see [Retrieve custom metrics with collectd \(p. 563\)](#).
- `ethtool` – Optional. Specifies that you want to retrieve network metrics using the `ethtool` plugin. This plugin can import both the metrics collected by the standard `ethtool` utility, as well as network performance metrics from Amazon EC2 instances. For more information about the configuration options available for `ethtool`, see [Collect network performance metrics \(p. 549\)](#).
- `cpu` – Optional. Specifies that CPU metrics are to be collected. This section is valid only for Linux instances. You must include at least one of the `resources` and `totalcpu` fields for any CPU metrics to be collected. This section can include the following fields:
 - `resources` – Optional. Specify this field with a value of `*` to cause per-cpu metrics are to be collected. The only allowed value is `*`.
 - `totalcpu` – Optional. Specifies whether to report cpu metrics aggregated across all cpu cores. The default is true.
 - `measurement` – Specifies the array of cpu metrics to be collected. Possible values are `time_active`, `time_guest`, `time_guest_nice`, `time_idle`, `time_iowait`, `time_irq`, `time_nice`, `time_softirq`, `time_steal`, `time_system`, `time_user`, `usage_active`, `usage_guest`, `usage_guest_nice`, `usage_idle`, `usage_iowait`, `usage_irq`, `usage_nice`, `usage_softirq`, `usage_steal`, `usage_system`, and `usage_user`. This field is required if you include `cpu`.

By default, the unit for `cpu_usage_*` metrics is `Percent`, and `cpu_time_*` metrics don't have a unit.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.

- **unit** – Specifies the unit to use for this metric, overriding the default unit of `None` or `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the [Unit description in MetricDatum](#).
- **metrics_collection_interval** – Optional. Specifies how often to collect the cpu metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds. For example, specifying 10 sets metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

- **append_dimensions** – Optional. Additional dimensions to use for only the cpu metrics. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics that the agent collects.
- **disk** – Optional. Specifies that disk metrics are to be collected. This section is valid only for Linux instances. This section can include as many as two fields:
 - **resources** – Optional. Specifies an array of disk mount points. This field limits CloudWatch to collect metrics from only the listed mount points. You can specify `*` as the value to collect metrics from all mount points. The default value is to collect metrics from all mount points.
 - **measurement** – Specifies the array of disk metrics to be collected. Possible values are `free`, `total`, `used`, `used_percent`, `inodes_free`, `inodes_used`, and `inodes_total`. This field is required if you include `disk`.

Note

The `disk` metrics have a dimension for `Partition`, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.

To see the default units for each `disk` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances \(p. 575\)](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- **rename** – Specifies a different name for this metric.
- **unit** – Specifies the unit to use for this metric, overriding the default unit of `None` or `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the [Unit description in MetricDatum](#).
- **ignore_file_system_types** – Specifies file system types to exclude when collecting disk metrics. Valid values include `sysfs`, `devtmpfs`, and so on.
- **drop_device** – Setting this to `true` causes `Device` to not be included as a dimension for disk metrics.

Preventing `Device` from being used as a dimension can be useful on instances that use the Nitro system because on those instances the device names change for each disk mount when the instance is rebooted. This can cause inconsistent data in your metrics and cause alarms based on these metrics to go to `INSUFFICIENT DATA` state.

The default is `false`.

- **metrics_collection_interval** – Optional. Specifies how often to collect the disk metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the disk metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `diskio` – Optional. Specifies that disk i/o metrics are to be collected. This section is valid only for Linux instances. This section can include as many as two fields:
 - `resources` – Optional. If you specify an array of devices, CloudWatch collects metrics from only those devices. Otherwise, metrics for all devices are collected. You can also specify * as the value to collect metrics from all devices.
 - `measurement` – Specifies the array of diskio metrics to be collected. Possible values are `reads`, `writes`, `read_bytes`, `write_bytes`, `read_time`, `write_time`, `io_time`, and `iops_in_progress`. This field is required if you include `diskio`.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the diskio metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the diskio metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `swap` – Optional. Specifies that swap memory metrics are to be collected. This section is valid only for Linux instances. This section can include as many as three fields:
 - `measurement` – Specifies the array of swap metrics to be collected. Possible values are `free`, `used`, and `used_percent`. This field is required if you include `swap`.

To see the default units for each swap metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances \(p. 575\)](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the swap metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the swap metrics. If you specify this field, it is used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent. It's collected as a high-resolution metric.
- `mem` – Optional. Specifies that memory metrics are to be collected. This section is valid only for Linux instances. This section can include as many as three fields:
 - `measurement` – Specifies the array of memory metrics to be collected. Possible values are `active`, `available`, `available_percent`, `buffered`, `cached`, `free`, `inactive`, `total`, `used`, and `used_percent`. This field is required if you include `mem`.

To see the default units for each `mem` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances \(p. 575\)](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the `mem` metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the `mem` metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics that the agent collects.
- `net` – Optional. Specifies that networking metrics are to be collected. This section is valid only for Linux instances. This section can include as many as four fields:
 - `resources` – Optional. If you specify an array of network interfaces, CloudWatch collects metrics from only those interfaces. Otherwise, metrics for all devices are collected. You can also specify `*` as the value to collect metrics from all interfaces.
 - `measurement` – Specifies the array of networking metrics to be collected. Possible values are `bytes_sent`, `bytes_recv`, `drop_in`, `drop_out`, `err_in`, `err_out`, `packets_sent`, and `packets_recv`. This field is required if you include `net`.

To see the default units for each `net` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances \(p. 575\)](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the `net` metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds. For example, specifying 10 sets metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the net metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `netstat` – Optional. Specifies that TCP connection state and UDP connection metrics are to be collected. This section is valid only for Linux instances. This section can include as many as three fields:
 - `measurement` – Specifies the array of netstat metrics to be collected. Possible values are `tcp_close`, `tcp_close_wait`, `tcp_closing`, `tcp_established`, `tcp_fin_wait1`, `tcp_fin_wait2`, `tcp_last_ack`, `tcp_listen`, `tcp_none`, `tcp_syn_sent`, `tcp_syn_recv`, `tcp_time_wait`, and `udp_socket`. This field is required if you include `netstat`.

To see the default units for each `netstat` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances \(p. 575\)](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the netstat metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the netstat metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `processes` – Optional. Specifies that process metrics are to be collected. This section is valid only for Linux instances. This section can include as many as three fields:
 - `measurement` – Specifies the array of processes metrics to be collected. Possible values are `blocked`, `dead`, `idle`, `paging`, `running`, `sleeping`, `stopped`, `total`, `total_threads`, `wait`, and `zombies`. This field is required if you include `processes`.

For all `processes` metrics, the default unit is `Count`.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the processes metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds. For example, specifying 10 sets metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the process metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
 - `nvidia_gpu` – Optional. Specifies that NVIDIA GPU metrics are to be collected. This section is valid only for Linux instances on hosts that are configured with a NVIDIA GPU accelerator and have the NVIDIA System Management Interface (`nvidia-smi`) installed.

The NVIDIA GPU metrics that are collected are prefixed with the string `nvidia_smi_` to distinguish them from the metrics collected for other accelerator types. This section can include as many as two fields:

- measurement – Specifies the array of NVIDIA GPU metrics to be collected. For a list of the possible values to use here, see the Metric column in the table in [Collect NVIDIA GPU metrics \(p. 552\)](#).

Within the entry for each individual metric, you can optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
 - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
 - `metrics_collection_interval` – Optional. Specifies how often to collect the NVIDIA GPU metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

`procstat` – Optional. Specifies that you want to retrieve metrics from individual processes. For more information about the configuration options available for `procstat`, see [Collect process metrics with the procstat plugin \(p. 554\)](#).

`statsd` – Optional. Specifies that you want to retrieve custom metrics using the `StatsD` protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard `StatsD` client to send the metrics to the CloudWatch agent. For more information about the configuration options available for `StatsD`, see [Retrieve custom metrics with StatsD \(p. 562\)](#).

`ethtool` – Optional. Specifies that you want to import `ethtool` statistics into CloudWatch. For more information, see [Collect network performance metrics \(p. 549\)](#).

The following is an example of a `metrics` section for a Linux server. In this example, three CPU metrics, three netstat metrics, three process metrics, and one disk metric are collected, and the agent is set up to receive additional metrics from a `collectd` client.

```
"metrics": {
  "metrics_collected": {
    "collectd": {},
    "cpu": {
      "resources": [
        "*"
      ],
      "measurement": [
        {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
        {"name": "cpu_usage_nice", "unit": "Percent"},
        "cpu_usage_guest"
      ],
      "totalcpu": false,
      "metrics_collection_interval": 10,
      "append_dimensions": {
        "test": "test1",
        "date": "2017-10-01"
      }
    }
  }
}.
```

```
    "netstat": {
        "measurement": [
            "tcp_established",
            "tcp_syn_sent",
            "tcp_close"
        ],
        "metrics_collection_interval": 60
    },
    "disk": {
        "measurement": [
            "used_percent"
        ],
        "resources": [
            "*"
        ],
        "drop_device": true
    },
    "processes": {
        "measurement": [
            "running",
            "sleeping",
            "dead"
        ]
    }
},
"append_dimensions": {
    "ImageId": "${aws:ImageId}",
    "InstanceId": "${aws:InstanceId}",
    "InstanceType": "${aws:InstanceType}",
    "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
"aggregation_dimensions" : [[ "AutoScalingGroupName"], [ "InstanceId", "InstanceType"]],
[]]
}
```

Windows Server

In the `metrics_collected` section for Windows Server, you can have subsections for each Windows performance object, such as `Memory`, `Processor`, and `LogicalDisk`. For information about what objects and counters are available, see the Microsoft Windows documentation.

Within the subsection for each object, you specify a `measurement` array of the counters to collect. The `measurement` array is required for each object that you specify in the configuration file. You can also specify a `resources` field to name the instances to collect metrics from. You can also specify `*` for `resources` to collect separate metrics for every instance. If you omit `resources`, the data for all instances is aggregated into one set. For objects that don't have instances, omit `resources`.

Within each object section, you can also specify the following optional fields:

- `metrics_collection_interval` – Optional. Specifies how often to collect the metrics for this object, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds. For example, specifying 10 sets metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics \(p. 95\)](#).

- `append_dimensions` – Optional. Specifies additional dimensions to use for only the metrics for this object. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent.

Within each counter section, you can also specify the following optional fields:

- **rename** – Specifies a different name to be used in CloudWatch for this metric.
- **unit** – Specifies the unit to use for this metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).

There are two other optional sections that you can include in `metrics_collected`:

- **statsd** – Enables you to retrieve custom metrics using the StatsD protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard StatsD client to send the metrics to the CloudWatch agent. For more information, see [Retrieve custom metrics with StatsD \(p. 562\)](#).
- **procstat** – Enables you to retrieve metrics from individual processes. For more information, see [Collect process metrics with the procstat plugin \(p. 554\)](#).

The following is an example `metrics` section for use on Windows Server. In this example, many Windows metrics are collected, and the computer is also set to receive additional metrics from a StatsD client.

```
"metrics": {  
    "metrics_collected": {  
        "statsd": {},  
        "Processor": {  
            "measurement": [  
                {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},  
                "% Interrupt Time",  
                "% User Time",  
                "% Processor Time"  
            ],  
            "resources": [  
                "*"  
            ],  
            "append_dimensions": {  
                "d1": "win_foo",  
                "d2": "win_bar"  
            }  
        },  
        "LogicalDisk": {  
            "measurement": [  
                {"name": "% Idle Time", "unit": "Percent"},  
                {"name": "% Disk Read Time", "rename": "DISK_READ"},  
                "% Disk Write Time"  
            ],  
            "resources": [  
                "*"  
            ]  
        },  
        "Memory": {  
            "metrics_collection_interval": 5,  
            "measurement": [  
                "Available Bytes",  
                "Cache Faults/sec",  
                "Page Faults/sec",  
                "Pages/sec"  
            ],  
            "append_dimensions": {  
                "d3": "win_bo"  
            }  
        },  
        "Network Interface": {  
            "metrics_collection_interval": 5,  
            "measurement": [  
                "Received Packets/sec",  
                "Transmitted Packets/sec",  
                "Received Bytes/sec",  
                "Transmitted Bytes/sec",  
                "Latency/  
            ]  
        }  
    }  
}
```

```
        "Bytes Received/sec",
        "Bytes Sent/sec",
        "Packets Received/sec",
        "Packets Sent/sec"
    ],
    "resources": [
        "*"
    ],
    "append_dimensions": {
        "d3": "win_bo"
    }
},
"System": {
    "measurement": [
        "Context Switches/sec",
        "System Calls/sec",
        "Processor Queue Length"
    ],
    "append_dimensions": {
        "d1": "win_foo",
        "d2": "win_bar"
    }
}
},
"append_dimensions": {
    "ImageId": "${aws:ImageId}",
    "InstanceId": "${aws:InstanceId}",
    "InstanceType": "${aws:InstanceType}",
    "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
"aggregation_dimensions" : [[{"ImageId"}, {"InstanceId", "InstanceType"}, {"d1"}],[]]
}
```

CloudWatch agent configuration file: Logs section

The logs section includes the following fields:

- `logs_collected` – Required if the logs section is included. Specifies which log files and Windows event logs are to be collected from the server. It can include two fields, `files` and `windows_events`.
 - `files` – Specifies which regular log files the CloudWatch agent is to collect. It contains one field, `collect_list`, which further defines these files.
 - `collect_list` – Required if `files` is included. Contains an array of entries, each of which specifies one log file to collect. Each of these entries can include the following fields:
 - `file_path` – Specifies the path of the log file to upload to CloudWatch Logs. Standard Unix glob matching rules are accepted, with the addition of `**` as a *super asterisk*. For example, specifying `/var/log/**.log` causes all `.log` files in the `/var/log` directory tree to be collected. For more examples, see [Glob Library](#).

You can also use the standard asterisk as a standard wildcard. For example, `/var/log/system.log*` matches files such as `system.log_1111`, `system.log_2222`, and so on in `/var/log`.

Only the latest file is pushed to CloudWatch Logs based on file modification time. We recommend that you use wildcards to specify a series of files of the same type, such as `access_log.2018-06-01-01` and `access_log.2018-06-01-02`, but not multiple kinds of files, such as `access_log_80` and `access_log_443`. To specify multiple kinds of files, add another log stream entry to the agent configuration file so that each kind of log file goes to a different log stream.

- `auto_removal` – Optional. If this is true, the CloudWatch agent automatically removes old log files after they are uploaded to CloudWatch Logs. The agent only removes complete files from logs that create multiple files, such as logs that create separate files for each date. If a log continuously writes to a single file, it is not removed.

If you already have a log file rotation or removal method in place, we recommend that you omit this field or set it to `false`.

If you omit this field, the default value of `false` is used.

- `log_group_name` – Optional. Specifies what to use as the log group name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

If you use these variables to create many different log groups, keep in mind the limit of 1,000,000 log groups per Region per account.

Allowed characters include a–z, A–Z, 0–9, '_' (underscore), '-' (hyphen), '/' (forward slash), and '.' (period).

We recommend that you specify this field to prevent confusion. If you omit this field, the file path up to the final dot is used as the log group name. For example, if the file path is `/tmp/TestLogFile.log.2017-07-11-14`, the log group name is `/tmp/TestLogFile.log`.

- `log_stream_name` – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

If you omit this field, the value of the `log_stream_name` parameter in the global logs section is used. If that is also omitted, the default value of `{instance_id}` is used.

If a log stream doesn't already exist, it's created automatically.

- `retention_in_days` – Optional. Specifies the number of days to retain the log events in the specified log group. If you omit this field for a log group that the agent is creating now, the retention will be set to never expire. If the agent is sending logs to a log group that already exists, and you omit this field, the log group's retention is not changed.

Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, and 3653.

If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

Note

The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies. For more information, see [Allowing the CloudWatch agent to set log retention policy \(p. 490\)](#).

Warning

If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- **filters** – Optional. Can contain an array of entries, each of which specifies a regular expression and a filter type to specify whether to publish or drop log entries that match the filter. If you omit this field, all logs in the log file are published to CloudWatch Logs. If you include this field, the agent processes each log message with all of the filters that you specify, and only the log events that pass all of the filters are published to CloudWatch Logs. The log entries that don't pass all of the filters will still remain in the host's log file, but will not be sent to CloudWatch Logs.

Each entry in the filters array can include the following fields:

- **type** – Denotes the type of filter. Valid values are `include` and `exclude`. With `include`, the log entry must match the expression to be published to CloudWatch Logs. With `exclude`, each log entry that matches the filter is not sent to CloudWatch Logs.
- **expression** – A regular expression string that follows the [RE2 Syntax](#).

Note

The CloudWatch agent doesn't check the performance of any regular expression that you supply, or restrict the run time of the evaluation of the regular expressions. We recommend that you are careful not to write an expression that is expensive to evaluate. For more information about possible issues, see [Regular expression Denial of Service - ReDoS](#)

For example, the following excerpt of the CloudWatch agent configuration file publishes logs that are PUT and POST requests to CloudWatch Logs, but excluding logs that come from Firefox.

```
"collect_list": [
    {
        "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
        "log_group_name": "test.log",
        "log_stream_name": "test.log",
        "filters": [
            {
                "type": "exclude",
                "expression": "Firefox"
            },
            {
                "type": "include",
                "expression": "P(UT|OST)"
            }
        ],
        ....
    }
]
```

Note

The order of the filters in the configuration file matters for performance. In the preceding example, the agent drops all the logs that match `Firefox` before it starts evaluating the second filter. To cause fewer log entries to be evaluated by more than one filter, put the filter that you expect to rule out more logs first in the configuration file.

- **timezone** – Optional. Specifies the time zone to use when putting timestamps on log events. The valid values are `UTC` and `Local`. The default value is `Local`.

This parameter is ignored if you don't specify a value for `timestamp_format`.

- **timestamp_format** – Optional. Specifies the timestamp format, using plaintext and special symbols that start with %. If you omit this field, the current time is used. If you use this field, you can use the symbols in the following list as part of the format.

If a single log entry contains two time stamps that match the format, the first time stamp is used.

This list of symbols is different than the list used by the older CloudWatch Logs agent. For a summary of these differences, see [Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent \(p. 596\)](#)

%y

Year without century as a zero-padded decimal number. For example, 19 to represent 2019.

%Y

Year with century as a decimal number. For example, 2019.

%b

Month as the locale's abbreviated name

%B

Month as the locale's full name

%m

Month as a zero-padded decimal number

%-m

Month as a decimal number (not zero-padded)

%d

Day of the month as a zero-padded decimal number

%-d

Day of the month as a decimal number (not zero-padded)

%A

Full name of weekday, such as Monday

%a

Abbreviation of weekday, such as Mon

%H

Hour (in a 24-hour clock) as a zero-padded decimal number

%I

Hour (in a 12-hour clock) as a zero-padded decimal number

%-I

Hour (in a 12-hour clock) as a decimal number (not zero-padded)

%p

AM or PM

%M

Minutes as a zero-padded decimal number

%-M

Minutes as a decimal number (not zero-padded)

`%S`

Seconds as a zero-padded decimal number

`%-S`

Seconds as a decimal number (not zero padded)

`%f`

Fractional seconds as a decimal number (1-9 digits), zero-padded on the left.

`%Z`

Time zone, for example `PST`

`%z`

Time zone, expressed as the offset between the local time zone and UTC. For example, `-0700`. Only this format is supported. For example, `-07:00` isn't a valid format.

- `multi_line_start_pattern` – Specifies the pattern for identifying the start of a log message. A log message is made of a line that matches the pattern and any subsequent lines that don't match the pattern.

If you omit this field, multi-line mode is disabled, and any line that begins with a non-whitespace character closes the previous log message and starts a new log message.

If you include this field, you can specify `{timestamp_format}` to use the same regular expression as your timestamp format. Otherwise, you can specify a different regular expression for CloudWatch Logs to use to determine the start lines of multi-line entries.

- `encoding` – Specified the encoding of the log file so that it can be read correctly. If you specify an incorrect coding, there might be data loss because characters that can't be decoded are replaced with other characters.

The default value is `utf-8`. The following are all possible values:

```
ascii, big5, euc-jp, euc-kr, gbk, gbk18030, ibm866, iso2022-jp,  
iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7,  
iso8859-8, iso8859-8-i, iso8859-10, iso8859-13, iso8859-14, iso8859-15,  
iso8859-16, koi8-r, koi8-u, macintosh, shift_jis, utf-8, utf-16,  
windows-874, windows-1250, windows-1251, windows-1252, windows-1253,  
windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, x-  
mac-cyrillic
```

- The `windows_events` section specifies the type of Windows events to collect from servers running Windows Server. It includes the following fields:
 - `collect_list` – Required if `windows_events` is included. Specifies the types and levels of Windows events to be collected. Each log to be collected has an entry in this section, which can include the following fields:
 - `event_name` – Specifies the type of Windows events to log. This is equivalent to the Windows event log channel name: for example, `System`, `Security`, `Application`, and so on. This field is required for each type of Windows event to log.

Note

When CloudWatch retrieves messages from a Windows log channel, it looks up the log channel based on its `Full Name` property. Meanwhile, the Windows Event Viewer navigation pane displays the `Log Name` property of log channels. The `Full Name` and `Log Name` do not always match. To confirm the `Full Name` of a channel, right-click on it in the Windows Event viewer and open **Properties**.

- **event_levels** – Specifies the levels of event to log. You must specify each level to log. Possible values include INFORMATION, WARNING, ERROR, CRITICAL, and VERBOSE. This field is required for each type of Windows event to log.
- **log_group_name** – Required. Specifies what to use as the log group name in CloudWatch Logs.
- **log_stream_name** – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use {instance_id}, {hostname}, {local_hostname}, and {ip_address} as variables within the name. {hostname} retrieves the hostname from the EC2 metadata, and {local_hostname} uses the hostname from the network configuration file.

If you omit this field, the value of the **log_stream_name** parameter in the global logs section is used. If that is also omitted, the default value of {instance_id} is used.

If a log stream doesn't already exist, it's created automatically.

- **event_format** – Optional. Specifies the format to use when storing Windows events in CloudWatch Logs. **xml** uses the XML format as in Windows Event Viewer. **text** uses the legacy CloudWatch Logs agent format.
- **log_stream_name** – Required. Specifies the default log stream name to be used for any logs or Windows events that don't have individual log stream names defined in the **log_stream_name** parameter within their entry in **collect_list**.
- **endpoint_override** – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends logs. Specifying this field and setting a private link enables you to send the logs to an Amazon VPC endpoint. For more information, see [What Is Amazon VPC?](#).

The value of **endpoint_override** must be a string that is a URL.

For example, the following part of the logs section of the configuration file sets the agent to use a VPC Endpoint when sending logs.

```
{  
  "logs": {  
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXX.logs.us-  
east-1.vpce.amazonaws.com",  
    .....  
  },  
}
```

- **force_flush_interval** – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5.
- **credentials** – Specifies an IAM role to use when sending logs to a different AWS account. If specified, this field contains one parameter, **role_arn**.
 - **role_arn** – Specifies the ARN of an IAM role to use for authentication when sending logs to a different AWS account. For more information, see [Sending metrics and logs to a different account \(p. 594\)](#). If specified here, this overrides the **role_arn** specified in the agent section of the configuration file, if any.
- **metrics_collected** – Specifies that the agent is to collect metrics embedded in logs. Currently, the **metrics_collected** field can contain only the **emf** field.
 - **emf** – Specifies that the agent is to collect logs that are in embedded metric format. You can generate metric data from these logs. For more information, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

The following is an example of a logs section.

```
"logs":
```

```
{  
    "logs_collected": {  
        "files": {  
            "collect_list": [  
                {  
                    "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-cloudwatch-agent.log",  
                    "log_group_name": "amazon-cloudwatch-agent.log",  
                    "log_stream_name": "my_log_stream_name_1",  
                    "timestamp_format": "%H: %M: %S%y%b%-d"  
                },  
                {  
                    "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\test.log",  
                    "log_group_name": "test.log",  
                    "log_stream_name": "my_log_stream_name_2"  
                }  
            ]  
        },  
        "windows_events": {  
            "collect_list": [  
                {  
                    "event_name": "System",  
                    "event_levels": [  
                        "INFORMATION",  
                        "ERROR"  
                    ],  
                    "log_group_name": "System",  
                    "log_stream_name": "System"  
                },  
                {  
                    "event_name": "CustomizedName",  
                    "event_levels": [  
                        "INFORMATION",  
                        "ERROR"  
                    ],  
                    "log_group_name": "CustomizedLogGroup",  
                    "log_stream_name": "CustomizedLogStream"  
                }  
            ]  
        },  
        "log_stream_name": "my_log_stream_name"  
    }  
}
```

How the CloudWatch agent handles sparse log files

Sparse files are files with both empty blocks and real contents. A sparse file uses disk space more efficiently by writing brief information representing the empty blocks to disk instead of the actual null bytes which make up the block. This makes the actual size of a sparse file usually much smaller than its apparent size.

However, the CloudWatch agent doesn't treat sparse files differently than it treats normal files. When the agent reads a sparse file, the empty blocks are treated as "real" blocks filled with null bytes. Because of this, the CloudWatch agent publishes as many bytes as the apparent size of a sparse file to CloudWatch.

Configuring the CloudWatch agent to publish a sparse file can cause higher than expected CloudWatch costs, so we recommend not to do so. For example, `/var/logs/lastlog` in Linux is usually a very sparse file, and we recommend that you don't publish it to CloudWatch.

CloudWatch agent configuration file: Complete examples

The following is an example of a complete CloudWatch agent configuration file for a Linux server.

The items listed in the measurement sections for the metrics you want to collect can either specify the complete metric name such as or just the part of the metric name that will be appended to the type of resource. For example, specifying either `reads` or `diskio_reads` in the measurement section of the `diskio` section will cause the `diskio_reads` metric to be collected.

This example includes both ways of specifying metrics in the measurement section.

```
{
  "agent": {
    "metrics_collection_interval": 10,
    "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log"
  },
  "metrics": {
    "namespace": "MyCustomNamespace",
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
          {"name": "cpu_usage_nice", "unit": "Percent"},
          "cpu_usage_guest"
        ],
        "totalcpu": false,
        "metrics_collection_interval": 10,
        "append_dimensions": {
          "customized_dimension_key_1": "customized_dimension_value_1",
          "customized_dimension_key_2": "customized_dimension_value_2"
        }
      },
      "disk": {
        "resources": [
          "/",
          "/tmp"
        ],
        "measurement": [
          {"name": "free", "rename": "DISK_FREE", "unit": "Gigabytes"},
          "total",
          "used"
        ],
        "ignore_file_system_types": [
          "sysfs", "devtmpfs"
        ],
        "metrics_collection_interval": 60,
        "append_dimensions": {
          "customized_dimension_key_3": "customized_dimension_value_3",
          "customized_dimension_key_4": "customized_dimension_value_4"
        }
      },
      "diskio": {
        "resources": [
          "*"
        ],
        "measurement": [
          "reads",
          "writes",
          "read_time",
          "write_time",
          "io_time"
        ],
        "metrics_collection_interval": 60
      },
      "swap": {
        "measurement": [

```

```
        "swap_used",
        "swap_free",
        "swap_used_percent"
    ],
},
"mem": {
    "measurement": [
        "mem_used",
        "mem_cached",
        "mem_total"
    ],
    "metrics_collection_interval": 1
},
"net": {
    "resources": [
        "eth0"
    ],
    "measurement": [
        "bytes_sent",
        "bytes_recv",
        "drop_in",
        "drop_out"
    ]
},
"netstat": {
    "measurement": [
        "tcp_established",
        "tcp_syn_sent",
        "tcp_close"
    ],
    "metrics_collection_interval": 60
},
"processes": {
    "measurement": [
        "running",
        "sleeping",
        "dead"
    ]
},
"append_dimensions": {
    "ImageId": "${aws:ImageId}",
    "InstanceId": "${aws:InstanceId}",
    "InstanceType": "${aws:InstanceType}",
    "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
"aggregation_dimensions" : [[{"ImageId"}, [{"InstanceId", "InstanceType"}, {"d1"}], []],
    "force_flush_interval" : 30
},
"logs": {
    "logs_collected": {
        "files": {
            "collect_list": [
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
                    "log_group_name": "amazon-cloudwatch-agent.log",
                    "log_stream_name": "amazon-cloudwatch-agent.log",
                    "timezone": "UTC"
                },
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
                    "log_group_name": "test.log",
                    "log_stream_name": "test.log",
                    "timezone": "Local"
                }
            ]
        }
    }
}
```

```
        }
    ]
}
},
"log_stream_name": "my_log_stream_name",
"force_flush_interval" : 15
}
}
```

The following is an example of a complete CloudWatch agent configuration file for a server running Windows Server.

```
{
  "agent": {
    "metrics_collection_interval": 60,
    "logfile": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-
cloudwatch-agent.log"
  },
  "metrics": {
    "namespace": "MyCustomNamespace",
    "metrics_collected": {
      "Processor": {
        "measurement": [
          {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
          "% Interrupt Time",
          "% User Time",
          "% Processor Time"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "customized_dimension_key_1": "customized_dimension_value_1",
          "customized_dimension_key_2": "customized_dimension_value_2"
        }
      },
      "LogicalDisk": {
        "measurement": [
          {"name": "% Idle Time", "unit": "Percent"},
          {"name": "% Disk Read Time", "rename": "DISK_READ"},
          "% Disk Write Time"
        ],
        "resources": [
          "*"
        ]
      },
      "customizedObjectName": {
        "metrics_collection_interval": 60,
        "customizedCounterName": [
          "metric1",
          "metric2"
        ],
        "resources": [
          "customizedInstances"
        ]
      },
      "Memory": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Available Bytes",
          "Cache Faults/sec",
          "Page Faults/sec",
          "Pages/sec"
        ]
      }
    }
  }
}
```

```
        },
        "Network Interface": {
            "metrics_collection_interval": 5,
            "measurement": [
                "Bytes Received/sec",
                "Bytes Sent/sec",
                "Packets Received/sec",
                "Packets Sent/sec"
            ],
            "resources": [
                "*"
            ],
            "append_dimensions": {
                "customized_dimension_key_3": "customized_dimension_value_3"
            }
        },
        "System": {
            "measurement": [
                "Context Switches/sec",
                "System Calls/sec",
                "Processor Queue Length"
            ]
        }
    },
    "append_dimensions": {
        "ImageId": "${aws:ImageId}",
        "InstanceId": "${aws:InstanceId}",
        "InstanceType": "${aws:InstanceType}",
        "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "aggregation_dimensions" : [[{"ImageId"}, [{"InstanceId", "InstanceType"}, {"d1"}], []]]
},
"logs": {
    "logs_collected": {
        "files": {
            "collect_list": [
                {
                    "file_path": "c:\\\\ProgramData\\\\Amazon\\\\AmazonCloudWatchAgent\\\\Logs\\\\amazon-cloudwatch-agent.log",
                    "log_group_name": "amazon-cloudwatch-agent.log",
                    "timezone": "UTC"
                },
                {
                    "file_path": "c:\\\\ProgramData\\\\Amazon\\\\AmazonCloudWatchAgent\\\\Logs\\\\test.log",
                    "log_group_name": "test.log",
                    "timezone": "Local"
                }
            ]
        },
        "windows_events": {
            "collect_list": [
                {
                    "event_name": "System",
                    "event_levels": [
                        "INFORMATION",
                        "ERROR"
                    ],
                    "log_group_name": "System",
                    "log_stream_name": "System",
                    "event_format": "xml"
                },
                {
                    "event_name": "CustomizedName",
                    "event_levels": [
                        "WARNING",

```

```
        "ERROR"
    ],
    "log_group_name": "CustomizedLogGroup",
    "log_stream_name": "CustomizedLogStream",
    "event_format": "xml"
}
}
},
"log_stream_name": "example_log_stream_name"
}
```

Save the CloudWatch agent configuration file manually

If you create or edit the CloudWatch agent configuration file manually, you can give it any name. For simplicity in troubleshooting, we recommend that you name it `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` on a Linux server and `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json` on servers running Windows Server. After you have created the file, you can copy it to other servers where you want to run the agent.

Uploading the CloudWatch agent configuration file to Systems Manager Parameter Store

If you plan to use the SSM Agent to install the CloudWatch agent on servers, after you manually edit the CloudWatch agent configuration file, you can upload it to Systems Manager Parameter Store. To do so, use the Systems Manager `put-parameter` command.

To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions. For more information, see [Create IAM roles and users for use with the CloudWatch agent \(p. 498\)](#).

Use the following command, where `parameter name` is the name to be used for this file in Parameter Store and `configuration_file_pathname` is the path and file name of the configuration file that you edited.

```
aws ssm put-parameter --name "parameter name" --type "String" --value file://configuration_file_pathname
```

Collect network performance metrics

EC2 instances running on Linux that use the Elastic Network Adapter (ENA) publish network performance metrics. Version 1.246396.0 and later of the CloudWatch agent enable you to import these network performance metrics into CloudWatch. When you import these network performance metrics into CloudWatch, they are charged as CloudWatch custom metrics.

For more information about the ENA driver, see [Enabling enhanced networking with the Elastic Network Adapter \(ENA\) on Linux instances](#) and [Enabling enhanced networking with the Elastic Network Adapter \(ENA\) on Windows instances](#).

How you set up the collection of network performance metrics differs on Linux servers and Windows servers.

The following table lists these network performance metrics enabled by the ENA adapter. When the CloudWatch agent imports these metrics into CloudWatch from Linux instances, it prepends `ethtool_` at the beginning of each of these metric names.

Metric	Description
Name on Linux servers: <code>bw_in_allowance_exceeded</code> Name on Windows servers: <code>Aggregate inbound BW allowance exceeded</code>	<p>The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance..</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
Name on Linux servers: <code>bw_out_allowance_exceeded</code> Name on Windows servers: <code>Aggregate outbound BW allowance exceeded</code>	<p>The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
Name on Linux servers: <code>conntrack_allowance_exceeded</code> Name on Windows servers: <code>Connection tracking allowance exceeded</code>	<p>The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
Name on Linux servers: <code>linklocal_allowance_exceeded</code> Name on Windows servers: <code>Link local packet rate allowance exceeded</code>	<p>The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
Name on Linux servers: <code>pps_allowance_exceeded</code> Name on Windows servers: <code>PPS allowance exceeded</code>	<p>The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code></p>

Metric	Description
	<p>section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>

Linux setup

On Linux servers, the *ethtool plugin* enables you to import the network performance metrics into CloudWatch.

ethtool is a standard Linux utility that can collect statistics about Ethernet devices on Linux servers. The statistics it collects depend on the network device and driver. Examples of these statistics include `tx_packets`, `rx_bytes`, `tx_errors`, and `align_errors`. When you use the ethtool plugin with the CloudWatch agent, you can also import these statistics into CloudWatch, along with the EC2 network performance metrics listed earlier in this section.

When the CloudWatch agent imports metrics into CloudWatch, it adds an `ethtool_` prefix to the names of all imported metrics. So the standard ethtool statistic `rx_bytes` is called `ethtool_rx_bytes` in CloudWatch, and the EC2 network performance metric `bw_in_allowance_exceeded` is called `ethtool_bw_in_allowance_exceeded` in CloudWatch.

On Linux servers, to import ethtool metrics, add an `ethtool` section to the `metrics_collected` section of the CloudWatch agent configuration file. The `ethtool` section can include the following subsections:

- **interface_include**— Including this section causes the agent to collect metrics from only the interfaces that have names listed in this section. If you omit this section, metrics are collected from all Ethernet interfaces that aren't listed in `interface_exclude`.

The default ethernet interface is `eth0`.

- **interface_exclude**— If you include this section, list the Ethernet interfaces that you don't want to collect metrics from.

The ethtool plugin always ignores loopback interfaces.

- **metrics_include**— This section lists the metrics to import into CloudWatch. It can include both standard statistics collected by ethtool and Amazon EC2 high-resolution network metrics.

The following example displays part of the CloudWatch agent configuration file. This configuration collects the standard ethtool metrics `rx_packets` and `tx_packets`, and the Amazon EC2 network performance metrics from only the `eth1` interface.

For more information about the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

```
"metrics": {
    "append_dimensions": {
        "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
        "ethtool": {
            "interface_include": [
                "eth1"
            ],
            "metrics_include": [
                "rx_packets",
                "tx_packets"
            ]
        }
    }
}
```

```
        "tx_packets",
        "bw_in_allowance_exceeded",
        "bw_out_allowance_exceeded",
        "conntrack_allowance_exceeded",
        "linklocal_allowance_exceeded",
        "pps_allowance_exceeded"
    ]
}
}
```

Viewing network performance metrics

After importing network performance metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view ethtool metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms \(p. 130\)](#).

Because all of these metrics are aggregate counters, you can use CloudWatch metric math functions such as RATE(METRICS()) to calculate the rate for these metrics in graphs or use them to set alarms. For more information about metric math functions, see [Using metric math \(p. 97\)](#).

To view network performance metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID, and then choose **Add to search**.
 - d. To filter by metric, choose the metric name, and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, and then choose **Add to dashboard**.

Collect NVIDIA GPU metrics

You can use the CloudWatch agent to collect NVIDIA GPU metrics from Linux servers. To set this up, add a nvidia_gpu section inside the metrics_collected section of the CloudWatch agent configuration file. For more information, see [Linux section \(p. 530\)](#).

The following metrics can be collected. All of these metrics are collected with no CloudWatch Unit, but you can specify a unit for each metric by adding a parameter to the CloudWatch agent configuration file. For more information, see [Linux section \(p. 530\)](#).

Metric	Metric name in CloudWatch	Description
utilization	nvidia_smi_utilization_gpu	The percentage of time over the past sample period during which one or more kernels on the GPU was running.

Metric	Metric name in CloudWatch	Description
temperature	nvidia_smi_temperature_gp	The core GPU temperature in degrees Celsius.
power_draw	nvidia_smi_power_draw	The last measured power draw for the entire board, in watts.
utilization	nvidia_smi_utilization_mem	The percentage of time over the past sample period during which global (device) memory was being read or written.
fan_speed	nvidia_smi_fan_speed	The percentage of maximum fan speed that the device's fan is currently intended to run at.
memory_total	nvidia_smi_memory_total	Reported total memory, in MB.
memory_used	nvidia_smi_memory_used	Memory used, in MB.
memory_free	nvidia_smi_memory_free	Memory free, in MB.
pcie_link_gen	nvidia_smi_pcie_link_gen	The current link generation.
pcie_link_width	nvidia_smi_pcie_link_width	The current link width.
encoder_stats	nvidia_smi_encoder_stats	The current number of encoder sessions.
encoder_stats	nvidia_smi_encoder_stats	The moving average of the encode frames per second.
encoder_stats	nvidia_smi_encoder_stats	The moving average of the encode latency in microseconds.
clocks_current	nvidia_smi_clocks_current	The current frequency of the graphics (shader) clock.
clocks_current	nvidia_smi_clocks_current	The current frequency of the Streaming Multiprocessor (SM) clock.
clocks_current	nvidia_smi_clocks_current	The current frequency of the memory clock.
clocks_current	nvidia_smi_clocks_current	The current frequency of the video (encoder plus decoder) clocks.

All of these metrics are collected with the following dimensions:

Dimension	Description
index	A unique identifier for the GPU on this server. Represents the NVIDIA Management Library (NVML) index of the device.
name	The type of GPU. For example,

Dimension	Description	
	NVIDIA Tesla A100	
host	The server host name.	

Collect process metrics with the procstat plugin

The *procstat* plugin enables you to collect metrics from individual processes. It is supported on Linux servers and on servers running Windows Server 2012 or later.

Topics

- [Configuring the CloudWatch agent for procstat \(p. 554\)](#)
- [Metrics collected by procstat \(p. 556\)](#)
- [Viewing process metrics imported by the CloudWatch agent \(p. 561\)](#)

Configuring the CloudWatch agent for procstat

To use the *procstat* plugin, add a *procstat* section in the *metrics_collected* section of the CloudWatch agent configuration file. There are three ways to specify the processes to monitor. You can use only one of these methods, but you can use that method to specify one or more processes to monitor.

- *pid_file*: Selects processes by the names of the process identification number (PID) files they create.
- *exe*: Selects the processes that have process names that match the string that you specify, using regular expression matching rules. The match is a "contains" match, meaning that if you specify *agent* as the term to match, processes with names like *cloudwatchagent* match the term. For more information, see [Syntax](#).
- *pattern*: Selects processes by the command lines used to start the processes. All processes are selected that have command lines matching the specified string using regular expression matching rules. The entire command line is checked, including parameters and options used with the command.

The match is a "contains" match, meaning that if you specify *-c* as the term to match, processes with parameters like *-config* match the term.

The CloudWatch agent uses only one of these methods, even if you include more than one of the above sections. If you specify more than one section, the CloudWatch agent uses the *pid_file* section if it is present. If not, it uses the *exe* section.

On Linux servers, the strings that you specify in an *exe* or *pattern* section are evaluated as regular expressions. On servers running Windows Server, these strings are evaluated as WMI queries. An example would be *pattern: "%apache%"*. For more information, see [LIKE Operator](#).

Whichever method you use, you can include an optional *metrics_collection_interval* parameter, which specifies how often in seconds to collect those metrics. If you omit this parameter, the default value of 60 seconds is used.

In the examples in the following sections, the *procstat* section is the only section included in the *metrics_collected* section of the agent configuration file. Actual configuration files can also include other sections in *metrics_collected*. For more information, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

Configuring with pid_file

The following example procstat section monitors the processes that create the PID files `example1.pid` and `example2.pid`. Different metrics are collected from each process. Metrics collected from the process that creates `example2.pid` are collected every 10 seconds, and the metrics collected from the `example1.pid` process are collected every 60 seconds, the default value.

```
{  
    "metrics": {  
        "metrics_collected": {  
            "procstat": [  
                {  
                    "pid_file": "/var/run/example1.pid",  
                    "measurement": [  
                        "cpu_usage",  
                        "memory_rss"  
                    ]  
                },  
                {  
                    "pid_file": "/var/run/example2.pid",  
                    "measurement": [  
                        "read_bytes",  
                        "read_count",  
                        "write_bytes"  
                    ],  
                    "metrics_collection_interval": 10  
                }  
            ]  
        }  
    }  
}
```

Configuring with exe

The following example procstat section monitors all processes with names that match the strings `agent` or `plugin`. The same metrics are collected from each process.

```
{  
    "metrics": {  
        "metrics_collected": {  
            "procstat": [  
                {  
                    "exe": "agent",  
                    "measurement": [  
                        "cpu_time",  
                        "cpu_time_system",  
                        "cpu_time_user"  
                    ]  
                },  
                {  
                    "exe": "plugin",  
                    "measurement": [  
                        "cpu_time",  
                        "cpu_time_system",  
                        "cpu_time_user"  
                    ]  
                }  
            ]  
        }  
    }  
}
```

Configuring with pattern

The following example procstat section monitors all processes with command lines that match the strings config or -c. The same metrics are collected from each process.

```
{  
    "metrics": {  
        "metrics_collected": {  
            "procstat": [  
                {  
                    "pattern": "config",  
                    "measurement": [  
                        "rlimit_memory_data_hard",  
                        "rlimit_memory_data_soft",  
                        "rlimit_memory_stack_hard",  
                        "rlimit_memory_stack_soft"  
                    ]  
                },  
                {  
                    "pattern": "-c",  
                    "measurement": [  
                        "rlimit_memory_data_hard",  
                        "rlimit_memory_data_soft",  
                        "rlimit_memory_stack_hard",  
                        "rlimit_memory_stack_soft"  
                    ]  
                }  
            ]  
        }  
    }  
}
```

Metrics collected by procstat

The following table lists the metrics that you can collect with the procstat plugin.

The CloudWatch agent adds procstat to the beginning of the following metric names. There is a different syntax depending on whether it was collected from a Linux server or a server running Windows Server. For example, the `cpu_time` metric appears as `procstat_cpu_time` when collected from Linux and as `procstat cpu_time` when collected from Windows Server.

Metric name	Available on	Description
<code>cpu_time</code>	Linux	The amount of time that the process uses the CPU. This metric is measured in hundredths of a second. Unit: Count
<code>cpu_time_system</code>	Linux, Windows Server, macOS	The amount of time that the process is in system mode. This metric is measured in hundredths of a second.

Metric name	Available on	Description
		Type: Float Unit: Count
cpu_time_user	Linux, Windows Server, macOS	The amount of time that the process is in user mode. This metric is measured in hundredths of a second. Unit: Count
cpu_usage	Linux, Windows Server, macOS	The percentage of time that the process is active in any capacity. Unit: Percent
memory_data	Linux, macOS	The amount of memory that the process uses for data. Unit: Bytes
memory_locked	Linux, macOS	The amount of memory that the process has locked. Unit: Bytes
memory_rss	Linux, Windows Server, macOS	The amount of real memory (resident set) that the process is using. Unit: Bytes
memory_stack	Linux, macOS	The amount of stack memory that the process is using. Unit: Bytes
memory_swap	Linux, macOS	The amount of swap memory that the process is using. Unit: Bytes

Metric name	Available on	Description
memory_vms	Linux, Windows Server, macOS	The amount of virtual memory that the process is using. Unit: Bytes
pid	Linux, Windows Server, macOS	Process identifier (ID). Unit: Count
pid_count	Linux, Windows Server, macOS	The number of process IDs associated with the process. On Linux servers and macOS computers the full name of this metric is <code>procstat_lookup_pid_count</code> and on Windows Server it is <code>procstat_lookup_pid_count</code> . Unit: Count
read_bytes	Linux, Windows Server	The number of bytes that the process has read from disks. Unit: Bytes
write_bytes	Linux, Windows Server	The number of bytes that the process has written to disks. Unit: Bytes
read_count	Linux, Windows Server	The number of disk read operations that the process has executed. Unit: Count

Metric name	Available on	Description
write_count	Linux, Windows Server	The number of disk write operations that the process has executed. Unit: Count
involuntary_context_switches	Linux	The number of times that the process was involuntarily context-switched. Unit: Count
voluntary_context_switches	Linux	The number of times that the process was context-switched voluntarily. Unit: Count
realtime_priority	Linux	The current usage of real-time priority for the process. Unit: Count
nice_priority	Linux	The current usage of nice priority for the process. Unit: Count
signals_pending	Linux	The number of signals pending to be handled by the process. Unit: Count
rlimit_cpu_time_hard	Linux	The hard CPU time resource limit for the process. Unit: Count

Metric name	Available on	Description
<code>rlimit_cpu_time_soft</code>	Linux	The soft CPU time resource limit for the process. Unit: Count
<code>rlimit_file_locks_hard</code>	Linux	The hard file locks resource limit for the process. Unit: Count
<code>rlimit_file_locks_soft</code>	Linux	The soft file locks resource limit for the process. Unit: Count
<code>rlimit_memory_data_hard</code>	Linux	The hard resource limit on the process for memory used for data. Unit: Bytes
<code>rlimit_memory_data_soft</code>	Linux	The soft resource limit on the process for memory used for data. Unit: Bytes
<code>rlimit_memory_locked_hard</code>	Linux	The hard resource limit on the process for locked memory. Unit: Bytes
<code>rlimit_memory_locked_soft</code>	Linux	The soft resource limit on the process for locked memory. Unit: Bytes

Metric name	Available on	Description
<code>rlimit_memory_rss_hard</code>	Linux	The hard resource limit on the process for physical memory. Unit: Bytes
<code>rlimit_memory_rss_soft</code>	Linux	The soft resource limit on the process for physical memory. Unit: Bytes
<code>rlimit_memory_stack_hard</code>	Linux	The hard resource limit on the process stack. Unit: Bytes
<code>rlimit_memory_stack_soft</code>	Linux	The soft resource limit on the process stack. Unit: Bytes
<code>rlimit_memory_vms_hard</code>	Linux	The hard resource limit on the process for virtual memory. Unit: Bytes

Viewing process metrics imported by the CloudWatch agent

After importing process metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view process metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms \(p. 130\)](#).

To view process metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:

- a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
- b. To sort the table, use the column heading.
- c. To filter by resource, choose the resource ID and then choose **Add to search**.
- d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

Retrieve custom metrics with StatsD

You can retrieve additional custom metrics from your applications or services using the CloudWatch agent with the StatsD protocol. StatsD is a popular open-source solution that can gather metrics from a wide variety of applications. StatsD is especially useful for instrumenting your own metrics. For an example of using the CloudWatch agent and StatsD together, see [How to better monitor your custom application metrics using Amazon CloudWatch Agent](#).

StatsD is supported on both Linux servers and servers running Windows Server. CloudWatch supports the following StatsD format:

```
MetricName:value|type|@sample_rate|#tag1:  
    value,tag1...
```

- MetricName – A string with no colons, bars, # characters, or @ characters.
- value – This can be either integer or float.
- type – Specify c for counter, g for gauge, ms for timer, h for histogram, or s for set.
- sample_rate – (Optional) A float between 0 and 1, inclusive. Use only for counter, histogram, and timer metrics. The default value is 1 (sampling 100% of the time).
- tags – (Optional) A comma-separated list of tags. StatsD tags are similar to dimensions in CloudWatch. Use colons for key/value tags, such as env:prod.

You can use any StatsD client that follows this format to send the metrics to the CloudWatch agent. For more information about some of the available StatsD clients, see the [StatsD client page on GitHub](#).

To collect these custom metrics, add a "statsd": {} line to the metrics_collected section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it's done for you. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#).

The StatsD default configuration works for most users. There are three optional fields that you can add to the statsd section of the agent configuration file as needed:

- service_address – The service address to which the CloudWatch agent should listen. The format is `ip:port`. If you omit the IP address, the agent listens on all available interfaces. Only the UDP format is supported, so you don't need to specify a UDP prefix.

The default value is :8125.

- metrics_collection_interval – How often in seconds that the StatsD plugin runs and collects metrics. The default value is 10 seconds. The range is 1–172,000.
- metrics_aggregation_interval – How often in seconds CloudWatch aggregates metrics into single data points. The default value is 60 seconds.

For example, if `metrics_collection_interval` is 10 and `metrics_aggregation_interval` is 60, CloudWatch collects data every 10 seconds. After each minute, the six data readings from that minute are aggregated into a single data point, which is sent to CloudWatch.

The range is 0–172,000. Setting `metrics_aggregation_interval` to 0 disables the aggregation of StatsD metrics.

The following is an example of the `statsd` section of the agent configuration file, using the default port and custom collection and aggregation intervals.

```
{  
    "metrics":{  
        "metrics_collected":{  
            "statsd":{  
                "service_address":":8125",  
                "metrics_collection_interval":60,  
                "metrics_aggregation_interval":300  
            }  
        }  
    }  
}
```

Viewing StatsD metrics imported by the CloudWatch agent

After importing StatsD metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view StatsD metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms \(p. 130\)](#).

To view StatsD metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

Retrieve custom metrics with collectd

You can retrieve additional metrics from your applications or services using the CloudWatch agent with the collectd protocol, which is supported only on Linux servers. collectd is a popular open-source solution with plugins that can gather system statistics for a wide variety of applications. By combining the system metrics that the CloudWatch agent can already collect with the additional metrics from collectd, you can better monitor, analyze, and troubleshoot your systems and applications. For more information about collectd, see [collectd - The system statistics collection daemon](#).

You use the collectd software to send the metrics to the CloudWatch agent. For the collectd metrics, the CloudWatch agent acts as the server while the collectd plugin acts as the client.

The collectd software is not installed automatically on every server. On a server running Amazon Linux 2, follow these steps to install collectd

```
sudo amazon-linux-extras install collectd
```

For information about installing collectd on other systems, see the [Download page for collectd](#).

To collect these custom metrics, add a "**collectd": {}**" line to the **metrics_collected** section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it is done for you. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#).

Optional parameters are also available. If you are using collectd and you do not use `/etc/collectd/auth_file` as your **collectd_auth_file**, you must set some of these options.

- **service_address:** The service address to which the CloudWatch agent should listen. The format is "udp://*ip:port*". The default is `udp://127.0.0.1:25826`.
- **name_prefix:** A prefix to attach to the beginning of the name of each collectd metric. The default is `collectd_`. The maximum length is 255 characters.
- **collectd_security_level:** Sets the security level for network communication. The default is `encrypt`.

encrypt specifies that only encrypted data is accepted. **sign** specifies that only signed and encrypted data is accepted. **none** specifies that all data is accepted. If you specify a value for **collectd_auth_file**, encrypted data is decrypted if possible.

For more information, see [Client setup](#) and [Possible interactions](#) in the collectd Wiki.

- **collectd_auth_file** Sets a file in which user names are mapped to passwords. These passwords are used to verify signatures and to decrypt encrypted network packets. If given, signed data is verified and encrypted packets are decrypted. Otherwise, signed data is accepted without checking the signature and encrypted data cannot be decrypted.

The default is `/etc/collectd/auth_file`.

If **collectd_security_level** is set to **none**, this is optional. If you set **collectd_security_level** to **encrypt** or **sign**, you must specify **collectd_auth_file**.

For the format of the auth file, each line is a user name followed by a colon and any number of spaces followed by the password. For example:

```
user1: user1_password
user2: user2_password
```

- **collectd_typesdb:** A list of one or more files that contain the dataset descriptions. The list must be surrounded by brackets, even if there is just one entry in the list. Each entry in the list must be surrounded by double quotes. If there are multiple entries, separate them with commas. The default on Linux servers is `["/usr/share/collectd/types.db"]`. The default on macOS computers depends on the version of collectd. For example, `["/usr/local/Cellar/collectd/5.12.0/share/collectd/types.db"]`.

For more information, see <https://collectd.org/documentation/manpages/types.db.5.shtml>.

- **metrics_aggregation_interval:** How often in seconds CloudWatch aggregates metrics into single data points. The default is 60 seconds. The range is 0 to 172,000. Setting it to 0 disables the aggregation of collectd metrics.

The following is an example of the collectd section of the agent configuration file.

```
{
  "metrics": {
```

```
"metrics_collected":{  
    "collectd":{  
        "name_prefix":"My_collectd_metrics_ ",  
        "metrics_aggregation_interval":120  
    }  
}
```

Viewing collectd metrics imported by the CloudWatch agent

After importing collectd metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view collectd metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms \(p. 130\)](#).

To view collectd metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

Set up and configure Prometheus metrics collection on Amazon EC2 instances

The following sections explain how to install the CloudWatch agent with Prometheus monitoring on EC2 instances, and how to configure the agent to scrape additional targets. It also provides tutorials for setting up sample workloads to use testing with Prometheus monitoring.

For information about the operating systems supported by the CloudWatch agent, see [Collecting metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent \(p. 482\)](#)

VPC security group requirements

If you are using a VPC, the following requirements apply.

- The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.
- The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Step 1: Install the CloudWatch agent \(p. 566\)](#)

- [Step 2: Scrape Prometheus sources and import metrics \(p. 566\)](#)
- [Example: Set up Java/JMX sample workloads for Prometheus metric testing \(p. 569\)](#)

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see [Installing the CloudWatch agent \(p. 483\)](#).

Step 2: Scrape Prometheus sources and import metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in `<scrape_config>` in the Prometheus documentation. The other is for the CloudWatch agent configuration.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in `<scrape_config>` in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. A sample configuration file contains the following global configuration lines:

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
- job_name: MY_JOB
  sample_limit: 10000
  file_sd_configs:
    - files: ["C:\\\\ProgramData\\\\Amazon\\\\AmazonCloudWatchAgent\\\\prometheus_sd_1.yaml", "C:\\\\ProgramData\\\\Amazon\\\\AmazonCloudWatchAgent\\\\prometheus_sd_2.yaml"]
```

The `global` section specifies parameters that are valid in all configuration contexts. They also serve as defaults for other configuration sections. It contains the following parameters:

- `scrape_interval`— Defines how frequently to scrape targets.
- `scrape_timeout`— Defines how long to wait before a scrape request times out.

The `scrape_configs` section specifies a set of targets and parameters that define how to scrape them. It contains the following parameters:

- `job_name`— The job name assigned to scraped metrics by default.
- `sample_limit`— Per-scrape limit on the number of scraped samples that will be accepted.
- `file_sd_configs`— List of file service discovery configurations. It reads a set of files containing a list of zero or more static configs. The `file_sd_configs` section contains a `files` parameter which defines patterns for files from which target groups are extracted.

The CloudWatch agent supports the following service discovery configuration types.

static_config Allows specifying a list of targets and a common label set for them. It is the canonical way to specify static targets in a scrape configuration.

The following is a sample static config to scrape Prometheus metrics from a local host. Metrics can also be scraped from other servers if the Prometheus port is open to the server where the agent runs.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_sd_1.yaml
- targets:
```

```
- 127.0.0.1:9404
labels:
  key1: value1
  key2: value2
```

This example contains the following parameters:

- **targets**— The targets scraped by the static config.
- **labels**— Labels assigned to all metrics that are scraped from the targets.

ec2_sd_config Allows retrieving scrape targets from Amazon EC2 instances. The following is a sample `ec2_sd_config` to scrape Prometheus metrics from a list of EC2 instances. The Prometheus ports of these instances have to open to the server where the CloudWatch agent runs. The IAM role for the EC2 instance where the CloudWatch agent runs must include the `ec2:DescribeInstance` permission. For example, you could attach the managed policy **AmazonEC2ReadOnlyAccess** to the instance running the CloudWatch agent.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: MY_JOB
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - i-98765432109876543
              - i-12345678901234567
```

This example contains the following parameters:

- **region**— The AWS Region where the target EC2 instance is. If you leave this blank, the Region from the instance metadata is used.
- **port**— The port to scrape metrics from.
- **filters**— Optional filters to use to filter the instance list. This example filters based on EC2 instance IDs. For more criteria that you can filter on, see [DescribeInstances](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file includes `prometheus` sections under both `logs` and `metrics_collected`. It includes the following parameters.

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path.
- **emf_processor**— specifies the embedded metric format processor configuration. For more information about embedded metric format, see [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format \(p. 761\)](#).

The `emf_processor` section can contain the following parameters:

- **metric_declaration_dedup**— It set to true, the de-duplication function for the embedded metric format metrics is enabled.

- **metric_namespace**— Specifies the metric namespace for the emitted CloudWatch metrics.
- **metric_unit**— Specifies the metric name:metric unit map. For information about supported metric units, see [MetricDatum](#).
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are metric_declaration sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
 - **source_labels** specifies the value of the labels that are checked by the **label_matcher** line.
 - **label_matcher** is a regular expression that checks the value of the labels listed in **source_labels**. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.
 - **metric_selectors** is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
 - **dimensions** is the list of labels to be used as CloudWatch dimensions for each selected metric.

The following is an example CloudWatch agent configuration for Prometheus.

```
{  
    "logs":{  
        "metrics_collected":{  
            "prometheus":{  
                "cluster_name": "prometheus-cluster",  
                "log_group_name": "Prometheus",  
                "prometheus_config_path": "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\  
\\prometheus.yaml",  
                "emf_processor":{  
                    "metric_declaration_dedup": true,  
                    "metric_namespace": "CWAgent-Prometheus",  
                    "metric_unit":{  
                        "jvm_threads_current": "Count",  
                        "jvm_gc_collection_seconds_sum": "Milliseconds"  
                    },  
                    "metric_declaration": [  
                        {  
                            "source_labels": [  
                                "job", "key2"  
                            ],  
                            "label_matcher": "MY_JOB;^value2",  
                            "dimensions": [  
                                [ "key1", "key2"  
                                ],  
                                [ "key2"  
                                ]  
                            ],  
                            "metric_selectors": [  
                                "^jvm_threads_current$",  
                                "^jvm_gc_collection_seconds_sum$"  
                            ]  
                        }  
                    ]  
                }  
            }  
        }  
    }  
}
```

The previous example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of the label `job` is `MY_JOB`
- The value of the label `key2` is `value2`
- The Prometheus metrics `jvm_threads_current` and `jvm_gc_collection_seconds_sum` contains both `job` and `key2` labels.

The log event that is sent includes the following highlighted section.

```
{  
    "CloudWatchMetrics": [  
        {  
            "Metrics": [  
                {  
                    "Unit": "Count",  
                    "Name": "jvm_threads_current"  
                },  
                {  
                    "Unit": "Milliseconds",  
                    "Name": "jvm_gc_collection_seconds_sum"  
                }  
            ],  
            "Dimensions": [  
                [  
                    "key1",  
                    "key2"  
                ],  
                [  
                    "key2"  
                ]  
            ],  
            "Namespace": "CWAgent-Prometheus"  
        }  
    ],  
    "ClusterName": "prometheus-cluster",  
    "InstanceId": "i-0e45bd06f196096c8",  
    "Timestamp": "1607966368109",  
    "Version": "0",  
    "host": "EC2AMAZ-PDDOIJUM",  
    "instance": "127.0.0.1:9404",  
    "jvm_threads_current": 2,  
    "jvm_gc_collection_seconds_sum": 0.00600000000000002,  
    "prom_metric_type": "gauge",  
    ...  
}
```

Example: Set up Java/JMX sample workloads for Prometheus metric testing

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see [prometheus/jmx_exporter](#).

The CloudWatch agent can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Hjava, and Tomcat (Catalina), from a JMX exporter on EC2 instances.

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see [Installing the CloudWatch agent \(p. 483\)](#).

Step 2: Start the Java/JMX workload

The next step is to start the Java/JMX workload.

First, download the latest JMX exporter jar file from the following location: [prometheus/jmx_exporter](#).

Use the jar for your sample application

The example commands in the following sections use `SampleJavaApplication-1.0-SNAPSHOT.jar` as the jar file. Replace these parts of the commands with the jar for your application.

Prepare the JMX exporter configuration

The `config.yaml` file is the JMX exporter configuration file. For more information, see [Configuration](#) in the JMX exporter documentation.

Here is a sample configuration for Java and Tomcat.

```
---
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=(\w+-\w+)-(\d+)\>"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
    help: Catalina global $3
    type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=/([-a-zA-Z0-9+\#@%?=~-|!:.,;]*[-a-zA-Z0-9+\#@%?=~-| ])><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
    help: Catalina servlet $3 total
    type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
    help: Catalina threadpool $3
    type: GAUGE

- pattern: 'Catalina<type=Manager, host=(-a-zA-Z0-9+\#@%?=~-|!:.,;)*[-a-zA-Z0-9+\#@%?=~-| ]><>(processingTime|sessionCounter|rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
    help: Catalina session $3 total
```

```
type: COUNTER
- pattern: "./*"
```

Start the Java application with the Prometheus exporter

Start the sample application. This will emit Prometheus metrics to port 9404. Be sure to replace the entry point `com.gubupt.sample.app.App` with the correct information for your sample java application.

On Linux, enter the following command.

```
$ nohup java -javaagent:./jmx_prometheus_javaagent-0.14.0.jar=9404:./config.yaml -cp ./SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App &
```

On Windows, enter the following command.

```
PS C:\> java -javaagent:.\jmx_prometheus_javaagent-0.14.0.jar=9404:.\config.yaml -cp .\SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

Verify the Prometheus metrics emission

Verify that Prometheus metrics are being emitted.

On Linux, enter the following command.

```
$ curl localhost:9404
```

On Windows, enter the following command.

```
PS C:\> curl http://localhost:9404
```

Sample output on Linux:

```
StatusCode      : 200
StatusDescription : OK
Content         : # HELP jvm_classes_loaded The number of classes that are currently
                  loaded in the JVM
                  # TYPE jvm_classes_loaded gauge
                  jvm_classes_loaded 2526.0
                  # HELP jvm_classes_loaded_total The total number of class...
RawContent      : HTTP/1.1 200 OK
                  Content-Length: 71908
                  Content-Type: text/plain; version=0.0.4; charset=utf-8
                  Date: Fri, 18 Dec 2020 16:38:10 GMT

                  # HELP jvm_classes_loaded The number of classes that are current...
Forms           : {}
Headers         : {[Content-Length, 71908], [Content-Type, text/plain; version=0.0.4;
                  charset=utf-8], [Date, Fri, 18
                  Dec 2020 16:38:10 GMT]}
Images          : {}
InputFields     : {}
Links           : {}
ParsedHtml      : System.__ComObject
RawContentLength : 71908
```

Step 3: Configure the CloudWatch agent to scrape Prometheus metrics

Next, set up the Prometheus scrape configuration in the CloudWatch agent configuration file.

To set up the Prometheus scrape configuration for the Java/JMX example

1. Set up the configuration for `file_sd_config` and `static_config`.

On Linux, enter the following command.

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml" ]
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "C:\\\\ProgramData\\\\Amazon\\\\AmazonCloudWatchAgent\\\\prometheus_file_sd.yaml" ]
```

2. Set up the scrape targets configuration.

On Linux, enter the following command.

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: linux
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

3. Set up the Prometheus scrape configuration by `ec2_sc_config`. Replace `your-ec2-instance-id` with the correct EC2 instance ID.

On Linux, enter the following command.

```
$ cat .\prometheus.yaml
global:
```

```
scrape_interval: 1m
scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - your-ec2-instance-id
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

- Set up the CloudWatch agent configuration. First, navigate to the correct directly. On Linux, it is `/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json`. On Windows, it is `C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json`.

The following is a sample configuration with Java/JHX Prometheus metrics defined. Be sure to replace `path-to-Prometheus-Scrape-Configuration-file` with the correct path.

```
{  
  "agent": {  
    "region": "us-east-1"  
  },  
  "logs": {  
    "metrics_collected": {  
      "prometheus": {  
        "cluster_name": "my-cluster",  
        "log_group_name": "prometheus-test",  
        "prometheus_config_path": "path-to-Prometheus-Scrape-Configuration-file",  
        "emf_processor": {  
          "metric_declaration_dedup": true,  
          "metric_namespace": "PrometheusTest",  
          "metric_unit": {  
            "jvm_threads_current": "Count",  
            "jvm_classes_loaded": "Count",  
            "java_lang_operatingsystem_freephysicalmemorysize": "Bytes",  
            "catalina_manager_activesessions": "Count",  
            "jvm_gc_collection_seconds_sum": "Seconds",  
            "catalina_globalrequestprocessor_bytesreceived": "Bytes",  
            "jvm_memory_bytes_used": "Bytes",  
            "jvm_memory_pool_bytes_used": "Bytes"  
          },  
          "metric_declaration": [  
            {  
              "source_labels": ["job"],  
              "label_matcher": "^jmx$",  
              "dimensions": [["instance"]],  
              "metric_selectors": [  
                "^jvm_threads_current$",
                "^jvm_classes_loaded$",
                "^java_lang_operatingsystem_freephysicalmemorysize$",
                "^catalina_manager_activesessions$",
                "^jvm_gc_collection_seconds_sum$"
              ]  
            }  
          ]  
        }  
      }  
    }  
  }  
}
```

```
        "catalina_globalrequestprocessor_bytesreceived$"
    ],
},
{
  "source_labels": ["job"],
  "label_matcher": "^jmx$",
  "dimensions": [["area"]],
  "metric_selectors": [
    "jvm_memory_bytes_used$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^jmx$",
  "dimensions": [["pool"]],
  "metric_selectors": [
    "jvm_memory_pool_bytes_used$"
  ]
}
],
},
"force_flush_interval": 5
}
```

5. Restart the CloudWatch agent by entering one of the following commands.

On Linux, enter the following command.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json
```

On Windows, enter the following command.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m ec2 -s -c file:C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json
```

Viewing the Prometheus metrics and logs

You can now view the Java/JMX metrics being collected.

To view the metrics for your sample Java/JMX workload

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **PrometheusTest** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **prometheus-test**.

Metrics collected by the CloudWatch agent

You can collect metrics from servers by installing the CloudWatch agent on the server. You can install the agent on both Amazon EC2 instances and on-premises servers, and on computers running either Linux, Windows Server, or macOS. If you install the agent on an Amazon EC2 instance, the metrics it collects are in addition to the metrics enabled by default on Amazon EC2 instances.

For information about installing the CloudWatch agent on an instance, see [Collecting metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent \(p. 482\)](#).

All metrics discussed in this section are collected directly by the CloudWatch agent.

Metrics collected by the CloudWatch agent on Windows Server instances

On a server running Windows Server, installing the CloudWatch agent enables you to collect the metrics associated with the counters in Windows Performance Monitor. The CloudWatch metric names for these counters are created by putting a space between the object name and the counter name. For example, the % Interrupt Time counter of the Processor object is given the metric name Processor % Interrupt Time in CloudWatch. For more information about Windows Performance Monitor counters, see the Microsoft Windows Server documentation.

The default namespace for metrics collected by the CloudWatch agent is CWAgent, although you can specify a different namespace when you configure the agent.

Metrics collected by the CloudWatch agent on Linux and macOS instances

The following table lists the metrics that you can collect with the CloudWatch agent on Linux servers and macOS computers.

Metric	Description
cpu_time_active	The amount of time that the CPU is active in any capacity. This metric is measured in hundredths of a second. Unit: None
cpu_time_guest	The amount of time that the CPU is running a virtual CPU for a guest operating system. This metric is measured in hundredths of a second. Unit: None
cpu_time_guest_nice	The amount of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. This metric is measured in hundredths of a second. Unit: None
cpu_time_idle	The amount of time that the CPU is idle. This metric is measured in hundredths of a second. Unit: None
cpu_time_iowait	The amount of time that the CPU is waiting for I/O operations to complete. This metric is measured in hundredths of a second. Unit: None

Metric	Description
<code>cpu_time_irq</code>	The amount of time that the CPU is servicing interrupts. This metric is measured in hundredths of a second. Unit: None
<code>cpu_time_nice</code>	The amount of time that the CPU is in user mode with low-priority processes, which can easily be interrupted by higher-priority processes. This metric is measured in hundredths of a second. Unit: None
<code>cpu_time_softirq</code>	The amount of time that the CPU is servicing software interrupts. This metric is measured in hundredths of a second. Unit: None
<code>cpu_time_steal</code>	The amount of time that the CPU is in <i>stolen time</i> , which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second. Unit: None
<code>cpu_time_system</code>	The amount of time that the CPU is in system mode. This metric is measured in hundredths of a second. Unit: None
<code>cpu_time_user</code>	The amount of time that the CPU is in user mode. This metric is measured in hundredths of a second. Unit: None
<code>cpu_usage_active</code>	The percentage of time that the CPU is active in any capacity. Unit: Percent
<code>cpu_usage_guest</code>	The percentage of time that the CPU is running a virtual CPU for a guest operating system. Unit: Percent
<code>cpu_usage_guest_nice</code>	The percentage of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. Unit: Percent
<code>cpu_usage_idle</code>	The percentage of time that the CPU is idle. Unit: Percent

Metric	Description
cpu_usage_iowait	The percentage of time that the CPU is waiting for I/O operations to complete. Unit: Percent
cpu_usage_irq	The percentage of time that the CPU is servicing interrupts. Unit: Percent
cpu_usage_nice	The percentage of time that the CPU is in user mode with low-priority processes, which higher-priority processes can easily interrupt. Unit: Percent
cpu_usage_softirq	The percentage of time that the CPU is servicing software interrupts. Unit: Percent
cpu_usage_steal	The percentage of time that the CPU is in <i>stolen time</i> , or time spent in other operating systems in a virtualized environment. Unit: Percent
cpu_usage_system	The percentage of time that the CPU is in system mode. Unit: Percent
cpu_usage_user	The percentage of time that the CPU is in user mode. Unit: Percent
disk_free	Free space on the disks. Unit: Bytes
disk_inodes_free	The number of available index nodes on the disk. Unit: Count
disk_inodes_total	The total number of index nodes reserved on the disk. Unit: Count
disk_inodes_used	The number of used index nodes on the disk. Unit: Count
disk_total	Total space on the disks, including used and free. Unit: Bytes
disk_used	Used space on the disks. Unit: Bytes

Metric	Description
disk_used_percent	The percentage of total disk space that is used. Unit: Percent
diskio_iops_in_progress	The number of I/O requests that have been issued to the device driver but have not yet completed. Unit: Count
diskio_io_time	The amount of time that the disk has had I/O requests queued. Unit: Milliseconds The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_reads	The number of disk read operations. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_read_bytes	The number of bytes read from the disks. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_read_time	The amount of time that read requests have waited on the disks. Multiple read requests waiting at the same time increase the number. For example, if 5 requests all wait for an average of 100 milliseconds, 500 is reported. Unit: Milliseconds The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_writes	The number disk write operations. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_write_bytes	The number of bytes written to the disks. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
diskio_write_time	<p>The amount of time that write requests have waited on the disks. Multiple write requests waiting at the same time increase the number. For example, if 8 requests all wait for an average of 1000 milliseconds, 8000 is reported.</p> <p>Unit: Milliseconds</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
ethtool_bw_in_allowance_exceeded	<p>The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance..</p> <p>This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
ethtool_bw_out_allowance_exceeded	<p>The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance..</p> <p>This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
ethtool_conntrack_allowance_exceeded	<p>The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance.</p> <p>This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>

Metric	Description
ethtool_linklocal_allowance_exceeded	<p>The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549)</p> <p>Unit: None</p>
ethtool_pps_allowance_exceeded	<p>The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics (p. 549).</p> <p>Unit: None</p>
mem_active	<p>The amount of memory that has been used in some way during the last sample period.</p> <p>Unit: Bytes</p>
mem_available	<p>The amount of memory that is available and can be given instantly to processes.</p> <p>Unit: Bytes</p>
mem_available_percent	<p>The percentage of memory that is available and can be given instantly to processes.</p> <p>Unit: Percent</p>
mem_buffered	<p>The amount of memory that is being used for buffers.</p> <p>Unit: Bytes</p>
mem_cached	<p>The amount of memory that is being used for file caches.</p> <p>Unit: Bytes</p>
mem_free	<p>The amount of memory that isn't being used.</p> <p>Unit: Bytes</p>
mem_inactive	<p>The amount of memory that hasn't been used in some way during the last sample period</p> <p>Unit: Bytes</p>

Metric	Description
mem_total	The total amount of memory. Unit: Bytes
mem_used	The amount of memory currently in use. Unit: Bytes
mem_used_percent	The percentage of memory currently in use. Unit: Percent
net_bytes_recv	The number of bytes received by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
net_bytes_sent	The number of bytes sent by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
net_drop_in	The number of packets received by this network interface that were dropped. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_drop_out	The number of packets transmitted by this network interface that were dropped. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_err_in	The number of receive errors detected by this network interface. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_err_out	The number of transmit errors detected by this network interface. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
net_packets_sent	<p>The number of packets sent by this network interface.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
net_packets_recv	<p>The number of packets received by this network interface.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
netstat_tcp_close	<p>The number of TCP connections with no state.</p> <p>Unit: Count</p>
netstat_tcp_close_wait	<p>The number of TCP connections waiting for a termination request from the client.</p> <p>Unit: Count</p>
netstat_tcp_closing	<p>The number of TCP connections that are waiting for a termination request with acknowledgement from the client.</p> <p>Unit: Count</p>
netstat_tcp_established	<p>The number of TCP connections established.</p> <p>Unit: Count</p>
netstat_tcp_fin_wait1	<p>The number of TCP connections in the FIN_WAIT1 state during the process of closing a connection.</p> <p>Unit: Count</p>
netstat_tcp_fin_wait2	<p>The number of TCP connections in the FIN_WAIT2 state during the process of closing a connection.</p> <p>Unit: Count</p>
netstat_tcp_last_ack	<p>The number of TCP connections waiting for the client to send acknowledgement of the connection termination message. This is the last state right before the connection is closed down.</p> <p>Unit: Count</p>
netstat_tcp_listen	<p>The number of TCP ports currently listening for a connection request.</p> <p>Unit: Count</p>
netstat_tcp_none	<p>The number of TCP connections with inactive clients.</p> <p>Unit: Count</p>

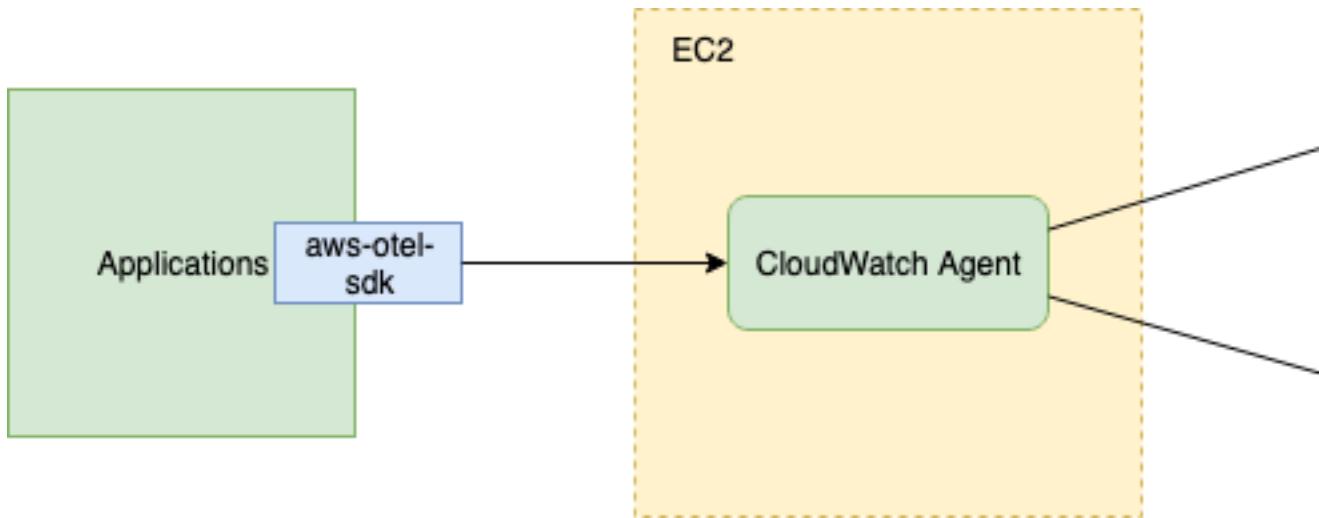
Metric	Description
netstat_tcp_syn_sent	The number of TCP connections waiting for a matching connection request after having sent a connection request. Unit: Count
netstat_tcp_syn_recv	The number of TCP connections waiting for connection request acknowledgement after having sent and received a connection request. Unit: Count
netstat_tcp_time_wait	The number of TCP connections currently waiting to ensure that the client received the acknowledgement of its connection termination request. Unit: Count
netstat_udp_socket	The number of current UDP connections. Unit: Count
processes_blocked	The number of processes that are blocked. Unit: Count
processes_dead	The number of processes that are dead, indicated by the X state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_idle	The number of processes that are idle (sleeping for more than 20 seconds). Available only on FreeBSD instances. Unit: Count
processes_paging	The number of processes that are paging, indicated by the w state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_running	The number of processes that are running, indicated by the R state code. Unit: Count
processes_sleeping	The number of processes that are sleeping, indicated by the S state code. Unit: Count

Metric	Description
<code>processes_stopped</code>	The number of processes that are stopped, indicated by the T state code. Unit: Count
<code>processes_total</code>	The total number of processes on the instance. Unit: Count
<code>processes_total_threads</code>	The total number of threads making up the processes. This metric is available only on Linux instances. This metric is not collected on macOS computers. Unit: Count
<code>processes_wait</code>	The number of processes that are paging, indicated by the w state code on FreeBSD instances. This metric is available only on FreeBSD instances, and is not available on Linux, Windows Server, or macOS instances. Unit: Count
<code>processes_zombies</code>	The number of zombie processes, indicated by the z state code. Unit: Count
<code>swap_free</code>	The amount of swap space that isn't being used. Unit: Bytes
<code>swap_used</code>	The amount of swap space currently in use. Unit: Bytes
<code>swap_used_percent</code>	The percentage of swap space currently in use. Unit: Percent

OpenTelemetry support in the CloudWatch agent

Versions 1.247347.3 and later of the CloudWatch agent have an embedded AWS OpenTelemetry Collector. This enables the CloudWatch agent to integrate with AWS OpenTelemetry APIs and SDKs, and send application telemetry data from EC2 instances to CloudWatch and AWS X-Ray. This feature is intended for existing CloudWatch agent users who want to begin monitoring with OpenTelemetry without installing or configuring multiple agents. For more information about AWS OpenTelemetry, see [AWS Distro for OpenTelemetry](#).

By using the CloudWatch agent with the embedded AWS OpenTelemetry Collector, you don't have to install a separate AWS OpenTelemetry Collector.



The CloudWatch agent with the embedded OpenTelemetry Collector can receive metrics and traces from the AWS OpenTelemetry SDK, and publish them to CloudWatch and X-Ray. The OpenTelemetry Collector that is embedded in the CloudWatch agent has the same behavior as the AWS OpenTelemetry Collector, and using it means that you don't need to install the separate AWS OpenTelemetry Collector. But if you do install both on the same server, be aware that the embedded OpenTelemetry Collector in the CloudWatch agent and the AWS OpenTelemetry Collector are located in different directories, managed through different tools, and run as separate processes. For example, if you configure and run both processes in the same server, be sure that the local ports that they use to listen do not conflict with each other.

OpenTelemetry in the CloudWatch agent is supported for the CloudWatch agent running on EC2 instances, but not for the CloudWatch agent running in containers or on on-premises servers. Both x86-64 and ARM64 architectures are supported on Linux instances, and x86-64 is supported on Windows Server instances.

IAM permissions

To be able to publish OpenTelemetry metrics and traces, the CloudWatch agent needs extra IAM permissions in addition to those permissions listed in the [CloudWatchAgentServerPolicy](#) managed policy. On servers where you want to use the agent's OpenTelemetry support, grant the following policy to the CloudWatch agent's IAM role that is attached to the instance. If you used the default suggested name in the documentation, then this role is called [CloudWatchAgentServerRole](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

}

CloudWatch agent OpenTelemetry configuration

Amazon provides a default configuration file for the embedded OpenTelemetry Collector. This configuration enables collecting OpenTelemetry metrics and traces through a default port. If the default configuration works for you, you don't need to do any more configuration steps. To see the the contents of this default configuration files, see [config.yaml](#) on Github.

If you want to customize the configuration file, see the [AWS Distro for OpenTelemetry](#) documentation. If you do this, once you have made the custom configuration file, you can either upload it to Parameter Store or copy it to the file system of every server where you want to use it. For more information about uploading it to Parameter Store, see [Uploading the CloudWatch agent configuration file to Systems Manager Parameter Store \(p. 549\)](#).

Use the command line to manage the CloudWatch agent with OpenTelemetry support

The CloudWatch agent with embedded OpenTelemetry support runs as two processes: the CloudWatch agent process and the OpenTelemetry Collector process. When you start the CloudWatch agent, each process determines separately whether it can start successfully.

You can run the `amazon-cloudwatch-agent-ctl` script to managed the OpenTelemetry Collector process. To configure and start the embedded OpenTelemetry Collector, run one of the the following commands:

Linux:

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a fetch-config -o configuration-file -s
```

Windows Server:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -o configuration-file -s
```

In this command, `-a fetch-config` loads the configuration for the OpenTelemetry Collector. The configuration can be the default configuration or a custom configuration from either Parameter Store or a local file. The `-s` parameter restarts the agent with this configuration.

The value of `configuration-file` can be `default` to use the default built-in configuration. To use a customized configuration in Parameter Store, you specify `ssm:your-configuration-parameter-name`. If you store the configuration in a local file instead, specify it as `file:your-configuration-file.yaml`.

For example, the following configures and starts the agent's OpenTelemetry process with the default built-in configuration for the OpenTelemetry Collector.

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a fetch-config -o default -s
```

Important

Use the `-o` option to configure the agent's OpenTelemetry Collector process, and use the `-c` option when you are configuring the CloudWatch agent process. The following command is an example that starts both the embedded OpenTelemetry Collector and the CloudWatch agent in Linux with each's default built-in configuration:

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a fetch-config -o default -c default -s
```

Use the command line to stop and start the OpenTelemetry Collector in the CloudWatch agent

Stop only the OpenTelemetry Collector process

To stop the OpenTelemetry process in the agent but let the CloudWatch agent process keep running, you use commands to tell the agent to remove the OpenTelemetry configuration, and then restart the agent with no configuration for the OpenTelemetry Collector.

In the following command, `-a remove-config -o` tells the CloudWatch agent to remove the configuration file for the OpenTelemetry Collector, and `-s` restarts the OpenTelemetry Collector without using any configuration, which effectively stops it.

Linux:

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a remove-config -o configuration-file -s
```

Windows Server:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a remove-config -o configuration-file -s
```

In this command, for `configuration-file` you can specify `all` to remove whatever configuration was currently applied. For example, the following command removes the configuration and stops the embedded OpenTelemetry Collector in Linux:

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a remove-config -o all -s
```

Stop both agent processes

You can use the `-a stop` parameter to stop both the embedded OpenTelemetry Collector process and the CloudWatch agent process if they are running.

Linux:

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a stop
```

Windows Server:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a stop
```

Start both agent processes

You can use the `-a start` parameter to start both the embedded OpenTelemetry Collector process and the CloudWatch agent process if they have already been configured. If you have never configured the OpenTelemetry Collector, using this parameter will not start it. If you have never started the CloudWatch agent process before, this command will start it with the default configuration.

Linux:

```
sudo /usr/bin/amazon-cloudwatch-agent-ctl -a start
```

Windows Server:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a start
```

Use Systems Manager to manage the CloudWatch agent with the embedded OpenTelemetry Collector

You can also use Systems Manager to manage the CloudWatch agent with the OpenTelemetry Collector.

To configure and start the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-
- If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **configure**.
7. In the **Optional OpenTelemetry Configuration Source** list, choose **ssm** if you want to use your custom configuration, or choose **default** to use the default configuration.
8. In the **Optional OpenTelemetry Configuration Location** box, enter the name of the OpenTelemetry configuration parameter that you created and saved to Systems Manager Parameter Store, as explained in the previous sections.
9. (Optional) If you want to configure and start the CloudWatch agent process at the same time, do the following:
 - a. In the **Optional Configuration Source** list, choose **ssm** if you want to use your custom configuration, or choose **default** to use the default configuration.
 - b. In the **Optional Configuration Location** box, enter the name of the CloudWatch agent configuration parameter that you created and saved to Systems Manager Parameter Store.
10. In the **Optional Restart** list, choose **yes** to start the agent after you have finished these steps.
11. Choose **Run**.
12. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully started.

To remove the configuration and stop the OpenTelemetry Collector process

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-
- If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **configure (remove)**.

7. In the **Optional OpenTelemetry Configuration Source** list, choose **all**.
8. In the **Optional Restart** list, choose **yes** to start the agent after you have finished these steps.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully started.

Generating OpenTelemetry metrics and traces

AWS provides OpenTelemetry SDKs and a Java auto instrumentation agent for your applications to use to generate OpenTelemetry metrics and traces and feed them to an OpenTelemetry Collector. For more information, see the following:

- [Getting Started with the Java SDK on Traces and Metrics Instrumentation](#)
- [Tracing with the AWS Distro for OpenTelemetry Go SDK](#)
- [Getting Started with the Python SDK](#)
- [Getting Started with the JavaScript SDK on Traces and Metrics Instrumentation](#)

Common scenarios with the CloudWatch agent

The following sections outline how to complete some common configuration and customization tasks when using the CloudWatch agent.

Topics

- [Running the CloudWatch agent as a different user \(p. 589\)](#)
- [Adding custom dimensions to metrics collected by the CloudWatch agent \(p. 591\)](#)
- [Multiple CloudWatch agent configuration files \(p. 591\)](#)
- [Aggregating or rolling up metrics collected by the CloudWatch agent \(p. 593\)](#)
- [Collecting high-resolution metrics with the CloudWatch agent \(p. 594\)](#)
- [Sending metrics and logs to a different account \(p. 594\)](#)
- [Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent \(p. 596\)](#)

Running the CloudWatch agent as a different user

On Linux servers, the CloudWatch runs as the root user by default. To have the agent run as a different user, use the `run_as_user` parameter in the `agent` section in the CloudWatch agent configuration file. This option is available only on Linux servers.

If you're already running the agent with the root user and want to change to using a different user, use one of the following procedures.

To run the CloudWatch agent as a different user on an EC2 instance running Linux

1. Download and install a new CloudWatch agent package. For more information, see [Download the CloudWatch agent package \(p. 484\)](#).
2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.
3. Provide credentials for this user in one of these ways:

- If the file `.aws/credentials` exists in the home directory of the root user, you must create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be `/home/username/ .aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information \(p. 496\)](#).
 - If the file `.aws/credentials` does not exist in the home directory of the root user, you can do one of the following:
 - Create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be `/home/username/ .aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information \(p. 496\)](#).
 - Instead of creating a credentials file, attach an IAM role to the instance. The agent uses this role as the credential provider.
4. In the CloudWatch agent configuration file, add the following line in the agent section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#)

5. Give the user necessary permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
6. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

To run the CloudWatch agent as a different user on an on-premises server running Linux

1. Download and install a new CloudWatch agent package. For more information, see [Download the CloudWatch agent package \(p. 484\)](#).
2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.
3. Store the credentials of this user to a path that the user can access, such as `/home/username/ .aws/credentials`.
4. Set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information \(p. 496\)](#).
5. In the CloudWatch agent configuration file, add the following line in the agent section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see [Create the CloudWatch agent configuration file \(p. 522\)](#)

6. Give the user necessary permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
7. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

Adding custom dimensions to metrics collected by the CloudWatch agent

To add custom dimensions such as tags to metrics collected by the agent, add the `append_dimensions` field to the section of the agent configuration file that lists those metrics.

For example, the following example section of the configuration file adds a custom dimension named `stackName` with a value of `Prod` to the `cpu` and `disk` metrics collected by the agent.

```
"cpu":{  
    "resources": [  
        "*"  
    ],  
    "measurement": [  
        "cpu_usage_guest",  
        "cpu_usage_nice",  
        "cpu_usage_idle"  
    ],  
    "totalcpu": false,  
    "append_dimensions": {  
        "stackName": "Prod"  
    }  
},  
"disk":{  
    "resources": [  
        "/",  
        "/tmp"  
    ],  
    "measurement": [  
        "total",  
        "used"  
    ],  
    "append_dimensions": {  
        "stackName": "Prod"  
    }  
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Multiple CloudWatch agent configuration files

You can set up the CloudWatch agent to use multiple configuration files. For example, you can use a common configuration file that collects a set of metrics and logs that you always want to collect from all servers in your infrastructure. You can then use additional configuration files that collect metrics from certain applications or in certain situations.

To set this up, first create the configuration files that you want to use. Any configuration files that will be used together on the same server must have different file names. You can store the configuration files on servers or in Parameter Store.

Start the CloudWatch agent using the `fetch-config` option and specify the first configuration file. To append the second configuration file to the running agent, use the same command but with the `append-config` option. All metrics and logs listed in either configuration file are collected. The following example Linux commands illustrate this scenario using configurations stores as files. The first line starts the agent using the `infrastructure.json` configuration file, and the second line appends the `app.json` configuration file.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s
-c file:/tmp/infrastructure.json
```

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -m ec2 -s
-c file:/tmp/app.json
```

The following example configuration files illustrate a use for this feature. The first configuration file is used for all servers in the infrastructure, and the second collects only logs from a certain application and is appended to servers running that application.

infrastructure.json

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          "usage_active"
        ],
        "totalcpu": true
      },
      "mem": {
        "measurement": [
          "used_percent"
        ]
      }
    },
    "logs": {
      "logs_collected": {
        "files": {
          "collect_list": [
            {
              "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
              "log_group_name": "amazon-cloudwatch-agent.log"
            },
            {
              "file_path": "/var/log/messages",
              "log_group_name": "/var/log/messages"
            }
          ]
        }
      }
    }
  }
}
```

app.json

```
{
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/app/app.log*",
            "log_group_name": "/app/app.log"
          }
        ]
      }
    }
  }
}
```

```
        }
    }
}
```

Any configuration files appended to the configuration must have different file names from each other and from the initial configuration file. If you use `append-config` with a configuration file with the same file name as a configuration file that the agent is already using, the `append` command overwrites the information from the first configuration file instead of appending to it. This is true even if the two configuration files with the same file name are on different file paths.

The preceding example shows the use of two configuration files, but there is no limit to the number of configuration files that you can append to the agent configuration. You can also mix the use of configuration files located on servers and configurations located in Parameter Store.

Aggregating or rolling up metrics collected by the CloudWatch agent

To aggregate or roll up metrics collected by the agent, add an `aggregation_dimensions` field to the section for that metric in the agent configuration file.

For example, the following configuration file snippet rolls up metrics on the `AutoScalingGroupName` dimension. The metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

```
"metrics": {
  "cpu": {...}
  "disk": {...}
  "aggregation_dimensions" : [[["AutoScalingGroupName"]]]
}
```

To roll up along the combination of each `InstanceId` and `InstanceType` dimensions in addition to rolling up on the Auto Scaling group name, add the following.

```
"metrics": {
  "cpu": {...}
  "disk": {...}
  "aggregation_dimensions" : [[["AutoScalingGroupName"], ["InstanceId", "InstanceType"]]]
}
```

To roll up metrics into one collection instead, use `[]`.

```
"metrics": {
  "cpu": {...}
  "disk": {...}
  "aggregation_dimensions" : [[]]
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Collecting high-resolution metrics with the CloudWatch agent

The `metrics_collection_interval` field specifies the time interval for the metrics collected, in seconds. By specifying a value of less than 60 for this field, the metrics are collected as high-resolution metrics.

For example, if your metrics should all be high-resolution and collected every 10 seconds, specify 10 as the value for `metrics_collection_interval` under the `agent` section as a global metrics collection interval.

```
"agent": {  
    "metrics_collection_interval": 10  
}
```

Alternatively, the following example sets the `cpu` metrics to be collected every second, and all other metrics are collected every minute.

```
"agent":{  
    "metrics_collection_interval": 60  
},  
"metrics":{  
    "metrics_collected":{  
        "cpu":{  
            "resources": [  
                "*"  
            ],  
            "measurement": [  
                "cpu_usage_guest"  
            ],  
            "totalcpu":false,  
            "metrics_collection_interval": 1  
        },  
        "disk":{  
            "resources": [  
                "/",  
                "/tmp"  
            ],  
            "measurement": [  
                "total",  
                "used"  
            ]  
        }  
    }  
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Sending metrics and logs to a different account

To have the CloudWatch agent send the metrics, logs, or both to a different account, specify a `role_arn` parameter in the agent configuration file on the sending server. The `role_arn` value specifies an IAM role in the target account that the agent uses when sending data to the target account. This role enables the sending account to assume a corresponding role in the target account when delivering the metrics or logs to the target account.

You can also specify two separate `role_arn` strings in the agent configuration file: one to use when sending metrics and another for sending logs.

The following example of part of the agent section of the configuration file sets the agent to use `CrossAccountAgentRole` when sending metrics and logs to a different account.

```
{
  "agent": {
    "credentials": {
      "role_arn": "arn:aws:iam::123456789012:role/CrossAccountAgentRole"
    }
  },
  ....
}
```

Alternatively, the following example sets different roles for the sending account to use for sending metrics and logs:

```
"metrics": {
  "credentials": {
    "role_arn": "RoleToSendMetrics"
  },
  "metrics_collected": {....}}
```

```
"logs": {
  "credentials": {
    "role_arn": "RoleToSendLogs"
  },
  ....}
```

Policies needed

When you specify a `role_arn` in the agent configuration file, you must also make sure the IAM roles of the sending and target accounts have certain policies. The roles in both the sending and target accounts should have `CloudWatchAgentServerPolicy`. For more information about assigning this policy to a role, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances \(p. 499\)](#).

The role in the sending account also must include the following policy. You add this policy on the **Permissions** tab in the IAM console when you edit the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::target-account-ID:role/agent-role-in-target-account"
      ]
    }
  ]
}
```

The role in the target account must include the following policy so that it recognizes the IAM role used by the sending account. You add this policy on the **Trust relationships** tab in the IAM console when you edit the role. The role in the target account where you add this policy is the role you created in [Create IAM roles and users for use with CloudWatch agent \(p. 489\)](#). This role is the role specified in `agent-role-in-target-account` in the policy used by the sending account.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "arn:aws:iam::sending-account-ID:role/role-in-sender-account"  
                ]  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent

The CloudWatch agent supports a different set of symbols for timestamp formats, compared to the earlier CloudWatch Logs agent. These differences are shown in the following table.

Symbols supported by both agents	Symbols supported only by unified CloudWatch agent	Symbols supported only by earlier CloudWatch Logs agent
%A, %a, %b, %B, %d, %f, %H, %l, %m, %M, %p, %S, %y, %Y, %Z, %z	%-d, %-l, %-m, %-M, %-S	%c,%j, %U, %W, %w

For more information about the meanings of the symbols supported by the new CloudWatch agent, see [CloudWatch Agent Configuration File: Logs Section](#) in the *Amazon CloudWatch User Guide*. For information about symbols supported by the CloudWatch Logs agent, see [Agent Configuration File](#) in the *Amazon CloudWatch Logs User Guide*.

Troubleshooting the CloudWatch agent

Use the following information to help troubleshoot problems with the CloudWatch agent.

Topics

- [CloudWatch agent command line parameters \(p. 597\)](#)
- [Installing the CloudWatch agent using Run Command fails \(p. 597\)](#)
- [The CloudWatch agent won't start \(p. 597\)](#)
- [Verify that the CloudWatch agent is running \(p. 597\)](#)
- [The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region \(p. 598\)](#)
- [The CloudWatch agent won't start on Windows Server \(p. 598\)](#)
- [Unable to find credentials on Windows Server \(p. 599\)](#)
- [Where are the metrics? \(p. 599\)](#)
- [I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console \(p. 599\)](#)

- [CloudWatch agent files and locations \(p. 599\)](#)
- [Finding information about CloudWatch agent versions \(p. 600\)](#)
- [Logs generated by the CloudWatch agent \(p. 601\)](#)
- [Stopping and restarting the CloudWatch agent \(p. 601\)](#)

CloudWatch agent command line parameters

To see the full list of parameters supported by the CloudWatch agent, enter the following at the command line at a computer where you have it installed:

```
amazon-cloudwatch-agent-ctl -help
```

Installing the CloudWatch agent using Run Command fails

To install the CloudWatch agent using Systems Manager Run Command, the SSM Agent on the target server must be version 2.2.93.0 or later. If your SSM Agent isn't the correct version, you might see errors that include the following messages:

```
no latest version found for package AmazonCloudWatchAgent on platform linux
```

```
failed to download installation package reliably
```

For information about updating your SSM Agent version, see [Installing and Configuring SSM Agent](#) in the [AWS Systems Manager User Guide](#).

The CloudWatch agent won't start

If the CloudWatch agent fails to start, there might be an issue in your configuration. Configuration information is logged in the `configuration-validation.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log` on servers running Windows Server.

Verify that the CloudWatch agent is running

You can query the CloudWatch agent to find whether it's running or stopped. You can use AWS Systems Manager to do this remotely. You can also use the command line, but only to check the local server.

To query the status of the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

3. If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
4. Choose **Run command**.
5. In the **Command document** list, choose the button next to **AmazonCloudWatch-ManageAgent**.

5. In the **Action** list, choose **status**.
6. For **Optional Configuration Source** choose **default** and keep **Optional Configuration Location** blank.
7. In the **Target** area, choose the instance to check.
8. Choose **Run**.

If the agent is running, the output resembles the following.

```
{  
    "status": "running",  
    "starttime": "2017-12-12T18:41:18",  
    "version": "1.73.4"  
}
```

If the agent is stopped, the "status" field displays "stopped".

To query the status of the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
```

On a server running Windows Server, enter the following in PowerShell as an administrator:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2  
-a status
```

The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region

If the agent doesn't start and the error message mentions an Amazon EC2 Region endpoint, you might have configured the agent to need access to the Amazon EC2 endpoint without granting that access.

For example, if you specify a value for the `append_dimensions` parameter in the agent configuration file that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see [Amazon Elastic Compute Cloud \(Amazon EC2\)](#) in the *Amazon Web Services General Reference*.

The CloudWatch agent won't start on Windows Server

On Windows Server, you might see the following error:

```
Start-Service : Service 'Amazon CloudWatch Agent (AmazonCloudWatchAgent)' cannot be started  
due to the following  
error:  
error: Cannot start service AmazonCloudWatchAgent on computer '.'.  
At C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1:113  
char:12  
+     $svc | Start-Service  
+     ~~~~~~  
+ CategoryInfo          : OpenError:  
(System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
```

```
ServiceCommandException
  + FullyQualifiedErrorId :
CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

To fix this, first make sure that the server service is running. This error can be seen if the agent tries to start when the server service isn't running.

If the server service is already running, the following may be the issue. On some Windows Server installations, the CloudWatch agent takes more than 30 seconds to start. Because Windows Server, by default, allows only 30 seconds for services to start, this causes the agent to fail with an error similar to the following:

To fix this issue, increase the service timeout value. For more information, see [A service does not start, and events 7000 and 7011 are logged in the Windows event log](#).

Unable to find credentials on Windows Server

On Windows Server, if you have credentials in a location other than `$SystemDrive\\Users\\Administrator\\.aws` on Windows Server 2012, or `"$SystemDrive\\Documents and Settings\\Administrator\\.aws` on Windows Server 2003, you can specify your own credential path by using the `shared_credential_file` option in `common.toml`.

If you don't have a credential file, you must create one. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information \(p. 496\)](#).

Where are the metrics?

If the CloudWatch agent has been running but you can't find metrics collected by it in the AWS Management Console or the AWS CLI, confirm that you're using the correct namespace. By default, the namespace for metrics collected by the agent is `CWAgent`. You can customize this namespace using the `namespace` field in the `metrics` section of the agent configuration file. If you don't see the metrics that you expect, check the configuration file to confirm the namespace being used.

When you first download the CloudWatch agent package, the agent configuration file is `amazon-cloudwatch-agent.json`. This file is in the directory where you ran the configuration wizard, or you might have moved it to a different directory. If you use the configuration wizard, the agent configuration file output from the wizard is named `config.json`. For more information about the configuration file, including the `namespace` field, see [CloudWatch agent configuration file: Metrics section \(p. 528\)](#).

I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console

If you update your CloudWatch agent configuration file, the next time that you start the agent, you need to use the `fetch-config` option. For example, if you stored the updated file on the local computer, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -s -m ec2 -c file:configuration-file-path
```

CloudWatch agent files and locations

The following table lists the files installed by and used with the CloudWatch agent, along with their locations on servers running Linux or Windows Server.

File	Linux location	Windows Server location
The control script that controls starting, stopping, and restarting the agent.	/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl or /usr/bin/amazon-cloudwatch-agent-ctl	\$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1
The log file the agent writes to. You might need to attach this when contacting AWS Support.	/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log or /var/log/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.log	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log
Agent configuration validation file.	/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log or /var/log/amazon/amazon-cloudwatch-agent/configuration-validation.log	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log
The JSON file used to configure the agent immediately after the wizard creates it. For more information, see Create the CloudWatch agent configuration file (p. 522) .	/opt/aws/amazon-cloudwatch-agent/bin/config.json	\$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\config.json
The JSON file used to configure the agent if this configuration file has been downloaded from Parameter Store.	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.json	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json
TOML file used to specify Region and credential information to be used by the agent, overriding system defaults.	/opt/aws/amazon-cloudwatch-agent/etc/common-config.toml or /etc/amazon/amazon-cloudwatch-agent/common-config.toml	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\common-config.toml

Finding information about CloudWatch agent versions

To find the version number of the CloudWatch agent on a Linux server, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
```

To find the version number of the CloudWatch agent on Windows Server, enter the following command:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2 -a  
status
```

Note

Using this command is the correct way to find the version of the CloudWatch agent. If you use **Programs and Features** in the Control Panel, you will see an incorrect version number.

You can also download a README file about the latest changes to the agent, and a file that indicates the version number that is currently available for download. These files are in the following locations:

- https://s3.amazonaws.com/amazoncloudwatch-agent/info/latest/RELEASE_NOTES or [https://s3.*region*.amazonaws.com/amazoncloudwatch-agent-*Region*/info/latest/RELEASE_NOTES](https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>Region</i>/info/latest/RELEASE_NOTES)
- https://s3.amazonaws.com/amazoncloudwatch-agent/info/latest/CWAGENT_VERSION or [https://s3.*region*.amazonaws.com/amazoncloudwatch-agent-*Region*/info/latest/CWAGENT_VERSION](https://s3.<i>region</i>.amazonaws.com/amazoncloudwatch-agent-<i>Region</i>/info/latest/CWAGENT_VERSION)

Logs generated by the CloudWatch agent

The agent generates a log while it runs. This log includes troubleshooting information. This log is the `amazon-cloudwatch-agent.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log` on servers running Windows Server.

You can configure the agent to log additional details in the `amazon-cloudwatch-agent.log` file. In the agent configuration file, in the agent section, set the `debug` field to `true`, then reconfigure and restart the CloudWatch agent. To disable the logging of this extra information, set the `debug` field to `false` and then restart the agent. For more information, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

In versions 1.247350.0 and later of the CloudWatch agent, you can optionally set the `aws_sdk_log_level` field in the agent section of the agent configuration file to one or more of the following options. Separate multiple options with the `|` character.

- `LogDebug`
- `LogDebugWithSigning`
- `LogDebugWithHTTPBody`
- `LogDebugRequestRetries`
- `LogDebugWithEventStreamBody`
- `LogDebug`

For more information about these options, see [LogLevelType](#).

Stopping and restarting the CloudWatch agent

You can manually stop the CloudWatch agent using either AWS Systems Manager or the command line.

To stop the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **stop**.
7. Keep **Optional Configuration Source** and **Optional Configuration Location** blank.
8. Choose **Run**.

To stop the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a stop
```

On a server running Windows Server, enter the following in PowerShell as an administrator:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2 -a stop
```

To restart the agent, follow the instructions in [Start the CloudWatch agent \(p. 506\)](#).

Amazon CloudWatch Application Insights

Amazon CloudWatch Application Insights facilitates observability for your applications and underlying AWS resources. It helps you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications. Application Insights, which is powered by [SageMaker](#) and other AWS technologies, provides automated dashboards that show potential problems with monitored applications, which help you to quickly isolate ongoing issues with your applications and infrastructure. The enhanced visibility into the health of your applications that Application Insights provides helps reduce mean time to repair (MTTR) to troubleshoot your application issues.

When you add your applications to Amazon CloudWatch Application Insights, it scans the resources in the applications and recommends and configures metrics and logs on [CloudWatch](#) for application components. Example application components include SQL Server backend databases and Microsoft IIS/Web tiers. Application Insights analyzes metric patterns using historical data to detect anomalies, and continuously detects errors and exceptions from your application, operating system, and infrastructure logs. It correlates these observations using a combination of classification algorithms and built-in rules. Then, it automatically creates dashboards that show the relevant observations and problem severity information to help you prioritize your actions. For common problems in .NET and SQL application stacks, such as application latency, SQL Server failed backups, memory leaks, large HTTP requests, and canceled I/O operations, it provides additional insights that point to a possible root cause and steps for resolution. Built-in integration with [AWS SSM OpsCenter](#) allows you to resolve issues by running the relevant Systems Manager Automation document.

Sections

- [What is Amazon CloudWatch Application Insights? \(p. 603\)](#)
- [How Amazon CloudWatch Application Insights works \(p. 608\)](#)
- [Get started with Amazon CloudWatch Application Insights \(p. 615\)](#)
- [Work with component configurations \(p. 634\)](#)
- [Create and configure CloudWatch Application Insights monitoring using CloudFormation templates \(p. 668\)](#)
- [Tutorial: Set up monitoring for SAP HANA \(p. 677\)](#)
- [Tutorial: Set up monitoring for .NET applications using SQL Server \(p. 685\)](#)
- [View and troubleshoot problems detected by Amazon CloudWatch Application Insights \(p. 690\)](#)
- [Logs and metrics supported by Amazon CloudWatch Application Insights \(p. 692\)](#)

What is Amazon CloudWatch Application Insights?

CloudWatch Application Insights helps you monitor your applications that use Amazon EC2 instances along with other [application resources \(p. 607\)](#). It identifies and sets up key metrics, logs, and alarms across your application resources and technology stack (for example, your Microsoft SQL Server database, web (IIS) and application servers, OS, load balancers, and queues). It continuously monitors metrics and logs to detect and correlate anomalies and errors. When errors and anomalies are detected, Application Insights generates [CloudWatch Events](#) that you can use to set up notifications or take actions. To assist with troubleshooting, it creates automated dashboards for detected problems, which include correlated metric anomalies and log errors, along with additional insights to point you to a potential root cause. The automated dashboards help you to take remedial actions to keep your applications healthy and to prevent impact to the end-users of your application. It also creates OpsItems so that you can resolve problems using [AWS SSM OpsCenter](#).

You can configure important counters, such as Mirrored Write Transaction/sec, Recovery Queue Length, and Transaction Delay, as well as Windows Event Logs on CloudWatch. When a failover event or problem occurs with your SQL HA workload, such as a restricted access to query a target database, CloudWatch Application Insights provides automated insights .

CloudWatch Application Insights integrates with [AWS Launch Wizard](#) to provide a one-click monitoring setup experience for deploying SQL Server HA workloads on AWS. When you select the option to set up monitoring and insights with Application Insights on the [Launch Wizard console](#), CloudWatch Application Insights automatically sets up relevant metrics, logs, and alarms on CloudWatch, and starts monitoring newly deployed workloads. You can view automated insights and detected problems, along with the health of your SQL Server HA workloads, on the CloudWatch console.

Contents

- [Features \(p. 604\)](#)
- [Concepts \(p. 604\)](#)
- [Pricing \(p. 605\)](#)
- [Related services \(p. 605\)](#)
- [Supported application components \(p. 607\)](#)
- [Supported technology stacks \(p. 608\)](#)

Features

Application Insights provides the following features.

Automatic set up of monitors for application resources

CloudWatch Application Insights reduces the time it takes to set up monitoring for your applications. It does this by scanning your application resources, providing a customizable list of recommended metrics and logs, and setting them up on CloudWatch to provide necessary visibility into your application resources, such as Amazon EC2 and Elastic Load Balancers (ELB). It also sets up dynamic alarms on monitored metrics. The alarms are automatically updated based on anomalies detected in the previous two weeks.

Problem detection and notification

CloudWatch Application Insights detects signs of potential problems with your application, such as metric anomalies and log errors. It correlates these observations to surface potential problems with your application. It then generates CloudWatch Events, [which can be configured to receive notifications or take actions \(p. 633\)](#). This eliminates the need for you to create individual alarms on metrics or log errors.

Troubleshooting

CloudWatch Application Insights creates CloudWatch automatic dashboards for problems that are detected. The dashboards show details about the problem, including the associated metric anomalies and log errors to help you with troubleshooting. They also provide additional insights that point to potential root causes of the anomalies and errors.

Concepts

The following concepts are important for understanding how Application Insights monitors your application.

Component

An auto-grouped, standalone, or custom grouping of similar resources that make up an application. We recommend grouping similar resources into custom components for better monitoring.

Observation

An individual event (metric anomaly, log error, or exception) that is detected with an application or application resource.

Problem

Problems are detected by correlating, classifying, and grouping related observations.

For definitions of other key concepts for CloudWatch Application Insights, see [Amazon CloudWatch Concepts](#).

Pricing

CloudWatch Application Insights sets up recommended metrics and logs for selected application resources using CloudWatch Metrics, Logs, and CloudWatch Events for notifications on detected problems. These features are charged to your AWS account according to [CloudWatch pricing](#). For the detected problems, it creates [CloudWatch Events](#) and [automatic dashboards](#). You are not charged for setup assistance, monitoring data analysis, or problem detection.

Related services

The following services are used along with CloudWatch Application Insights:

Related AWS services

- **Amazon CloudWatch** provides system-wide visibility into resource utilization, application performance, and operational health. It collects and tracks metrics, sends alarm notifications, automatically updates resources that you are monitoring based on the rules that you define, and allows you to monitor your own custom metrics. CloudWatch Application Insights is initiated through CloudWatch—specifically, within the CloudWatch default operational dashboards. For more information, see the [Amazon CloudWatch User Guide](#).
- **CloudWatch Container Insights** collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. You can use Container Insights to monitor Amazon ECS, Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2. When Application Insights is enabled on the Container Insights or Application Insights consoles, Application Insights displays detected problems on your Container Insights dashboard. For more information, see [Using Container Insights \(p. 305\)](#).
- **Amazon DynamoDB** is a fully managed NoSQL database service that lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.
- **Amazon EC2** provides scalable computing capacity in the AWS Cloud. You can use Amazon EC2 to launch as many or as few virtual servers as you need, to configure security and networking, and to manage storage. You can scale up or down to handle changes in requirements or spikes in popularity, which reduces your need to forecast traffic. For more information, see the [Amazon EC2 User Guide for Linux Instances](#) or [Amazon EC2 Guide for Windows Instances](#).
- **Amazon Elastic Block Store (Amazon EBS)** provides block-level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes behave like raw, unformatted block devices. You can mount these volumes as devices on your instances. Amazon EBS volumes that are attached to an instance are exposed as storage volumes that persist independently from the life of the instance. You can create a file system on top of these volumes, or use them in any way you would use a block device (such as

a hard drive). You can dynamically change the configuration of a volume attached to an instance. For more information, see the [Amazon EBS User Guide](#).

- **Amazon EC2 Auto Scaling** helps ensure that you have the correct number of EC2 instances available to handle the load for your application. For more information, see the [Amazon EC2 Auto Scaling User Guide](#).
- **Elastic Load Balancing** distributes incoming applications or network traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in multiple Availability Zones. For more information, see the [Elastic Load Balancing User Guide](#).
- **IAM** is a web service that helps you to securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication), and to control the resources they can use and how they can use them (authorization). For more information, see [Authentication and Access Control for Amazon CloudWatch](#).
- **AWS Lambda** lets you build serverless applications composed of functions that are triggered by events and automatically deploy them using CodePipeline and AWS CodeBuild. For more information, see [AWS Lambda Applications](#).
- **AWS Launch Wizard for SQL Server** reduces the time it takes to deploy SQL Server high availability solution to the cloud. You input your application requirements, including performance, number of nodes, and connectivity on the service console, and AWS Launch Wizard identifies the right AWS resources to deploy and run your SQL Server Always On application.
- **AWS Resource Groups** help you to organize the resources that make up your application. With Resource Groups, you can manage and automate tasks on a large number of resources at one time. Only one Resource Group can be registered for a single application. For more information, see the [AWS Resource Groups User Guide](#).
- **Amazon SQS** offers a secure, durable, and available hosted queue that allows you to integrate and decouple distributed software systems and components. For more information, see the [Amazon SQS User Guide](#).
- **AWS Step Functions** is a serverless function composer that allows you to sequence a variety of AWS services and resources, including AWS Lambda functions, into structured, visual workflows. For more information, see the [AWS Step Functions User Guide](#).
- **AWS SSM OpsCenter** aggregates and standardizes OpsItems across services while providing contextual investigation data about each OpsItem, related OpsItems, and related resources. OpsCenter also provides Systems Manager Automation documents (runbooks) that you can use to quickly resolve issues. You can specify searchable, custom data for each OpsItem. You can also view automatically-generated summary reports about OpsItems by status and source. For more information, see the [AWS Systems Manager User Guide](#).
- **Amazon API Gateway** is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. For more information, see the [Amazon API Gateway User Guide](#).

Note

Application Insights supports only REST API protocols (v1 of the API Gateway service).

- **Amazon Elastic Container Service (Amazon ECS)** is a fully managed container orchestration service. You can use Amazon ECS to run your most sensitive and mission-critical applications. For more information, see the [Amazon Elastic Container Service Developer Guide](#).
- **Amazon Elastic Kubernetes Service (Amazon EKS)** is a managed service that you can use to run Kubernetes on AWS without having to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. For more information, see the [Amazon EKS User Guide](#).
- **Kubernetes on Amazon EC2**. Kubernetes is open-source software that helps you deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs containers on those instances with processes for deployment, maintenance, and scaling. With Kubernetes you can run any type of containerized application with the same toolset on-premises and in the cloud. For more information, see [Running Kubernetes on AWS EC2](#).

- **Amazon FSx** helps you to launch and run popular file systems that are fully managed by AWS. With Amazon FSx, you can leverage the feature sets and performance of common open source and commercially-licensed file systems to avoid time-consuming administrative tasks. For more information, see the [Amazon FSx Documentation](#).
- **Amazon Simple Notification Service (SNS)** is a fully-managed messaging service for both application-to-application and application-to-person communication. You can configure Amazon SNS for monitoring by Application Insights. When Amazon SNS is configured as a resource for monitoring, Application Insights tracks SNS metrics to help determine why SNS messages may encounter issues or fail.

Related third-party services

- For some workloads and applications monitored in Application Insights, **Prometheus JMX exporter** is installed using AWS Systems Manager Distributor so that CloudWatch Application Insights can retrieve Java-specific metrics. When you choose to monitor a Java application, Application Insights automatically installs the Prometheus JMX exporter for you.

Supported application components

CloudWatch Application Insights scans your resource group to identify application components. Components can be standalone, auto-grouped (such as instances in an Auto Scaling group or behind a load balancer), or custom (by grouping together individual EC2 instances).

The following components are supported by CloudWatch Application Insights:

AWS components

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Elastic Load Balancing: Application Load Balancer and Classic Load Balancer (all target instances of these load balancers are identified and configured).
- Amazon EC2 Auto Scaling groups: AWS Auto Scaling (Auto Scaling groups are dynamically configured for all target instances; if your application scales up, CloudWatch Application Insights automatically configure the new instances). Auto Scaling groups are not supported for CloudFormation-based Resource Groups.
- AWS Lambda
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB table
- Amazon S3 bucket metrics
- AWS Step Functions
- Amazon API Gateway REST API stages
- Amazon Elastic Container Service (Amazon ECS): cluster, service, and task
- Amazon Elastic Kubernetes Service (Amazon EKS): cluster
- Kubernetes on Amazon EC2: Kubernetes cluster running on EC2
- Amazon SNS topic

Any other component type resources are not currently tracked by CloudWatch Application Insights. If a component type that is supported does not appear in your Application Insights application, the component may already be registered and managed by another application you own that is monitored by Application Insights.

Supported technology stacks

You can use CloudWatch Application Insights to monitor your applications running on Windows Server and Linux operating systems by selecting the application tier dropdown menu option for one of the following technologies:

- Front-end: Microsoft Internet Information Services (IIS) Web Server
- Worker-tier:
 - .NET Framework
 - .NET Core
- Applications: Java
- Active Directory
- SharePoint
- Databases:
 - Microsoft SQL Server running on Amazon RDS or Amazon EC2 (including SQL Server High Availability configurations. See, [Component configuration examples \(p. 640\)](#)).
 - MySQL running on Amazon RDS, Amazon Aurora, or Amazon EC2
 - PostgreSQL running on Amazon RDS or Amazon EC2
 - Amazon DynamoDB table
 - Oracle running on Amazon RDS or Amazon EC2
 - SAP HANA database on a single Amazon EC2 instance and multiple EC2 instances
 - Cross-AZ SAP HANA database high availability setup.

If none of the technology stacks listed above apply to your application resources, you can monitor your application stack by choosing **Custom** from the application tier dropdown menu on the **Manage monitoring** page.

How Amazon CloudWatch Application Insights works

This section contains information about how CloudWatch Application Insights works, including:

- [How Application Insights monitors applications \(p. 608\)](#)
- [Data retention \(p. 609\)](#)
- [Quotas \(p. 609\)](#)
- [AWS Systems Manager \(SSM\) packages used by CloudWatch Application Insights \(p. 609\)](#)

How Application Insights monitors applications

Application Insights monitors applications as follows.

Application discovery and configuration

The first time an application is added to CloudWatch Application Insights it scans the application components to recommend key metrics, logs, and other data sources to monitor for your application. You can then configure your application based on these recommendations.

Data preprocessing

CloudWatch Application Insights continuously analyzes the data sources being monitored across the application resources to discover metric anomalies and log errors (observations).

Intelligent problem detection

The CloudWatch Application Insights engine detects problems in your application by correlating observations using classification algorithms and built-in rules. To assist in troubleshooting, it creates automated CloudWatch dashboards, which include contextual information about the problems.

Alert and action

When CloudWatch Application Insights detects a problem with your application, it generates CloudWatch Events to notify you of the problem. See [Application Insights CloudWatch Events and notifications for detected problems \(p. 633\)](#) for more information about how to set up these Events.

Example scenario

You have an ASP .NET application that is backed by a SQL Server database. Suddenly, your database begins to malfunction because of high memory pressure. This leads to application performance degradation and possibly HTTP 500 errors in your web servers and load balancer.

With CloudWatch Application Insights and its intelligent analytics, you can identify the application layer that is causing the problem by checking the dynamically created dashboard that shows the related metrics and log file snippets. In this case, the problem might be at the SQL database layer.

Data retention

CloudWatch Application Insights retains problems for 55 days and observations for 60 days.

Quotas

For default quotas for CloudWatch Application Insights, see [Amazon CloudWatch Application Insights endpoints and quotas](#). Unless otherwise noted, each quota is per AWS Region. Contact [AWS Support](#) to request an increase in your service quota. Many services contain quotas that cannot be changed. For more information about the quotas for a specific service, see the documentation for that service.

AWS Systems Manager (SSM) packages used by CloudWatch Application Insights

The packages listed in this section are used by Application Insights, and can be independently managed and deployed with AWS Systems Manager Distributor. For more information about SSM Distributor, see [AWS Systems Manager Distributor](#) in the *AWS Systems Manager User Guide*.

Packages:

- [AWSObservabilityExporter-JMXExporterInstallAndConfigure \(p. 609\)](#)
- [AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure \(p. 612\)](#)
- [AWSObservabilityExporter-HAClusterExporterInstallAndConfigure \(p. 614\)](#)

AWSObservabilityExporter-JMXExporterInstallAndConfigure

You can retrieve workload-specific Java metrics from [Prometheus JMX exporter](#) for Application Insights to configure and monitor alarms. In the Application Insights console, on the **Manage monitoring** page,

select **JAVA application** from the **Application tier** dropdown. Then under **JAVA Prometheus exporter configuration**, select your **Collection method** and **JMX port number**.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the AWS-provided Prometheus JMX exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus JMX exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- The JAVA_HOME environment variable is set

Install and configure the `AWSObservabilityExporter-JMXExporterInstallAndConfigure` package

The `AWSObservabilityExporter-JMXExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure [Prometheus JMX Exporter](#). When Java metrics are sent by the Prometheus JMX exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the [Prometheus JMX exporter YAML configuration file](#) located in the Prometheus GitHub repository. Use the example configuration and option descriptions to guide you.
2. Copy the Prometheus JMX exporter YAML configuration file encoded as Base64 to a new SSM parameter in [SSM Parameter Store](#).
3. Navigate to the [SSM Distributor](#) console and open the **Third party** tab. Select `AWSObservabilityExporter-JMXExporterInstallAndConfigure` and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{  
  "SSM_EXPORTER_CONFIGURATION": "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",  
  "SSM_EXPOSITION_PORT": "9404"  
}
```

Note

Port 9404 is the default port used to send Prometheus JMX metrics. You can update this port.

Example: Configure CloudWatch agent to retrieve Java metrics

1. Install the Prometheus JMX exporter, as described in the previous procedure. Then verify that it is correctly installed on your instance by checking the port status.

Successful installation on Windows instance example

```
PS C:\> curl http://localhost:9404 (http://localhost:9404/)  
StatusCode : 200  
StatusDescription : OK  
Content : # HELP jvm_info JVM version info
```

Successful installation on Linux instance example

```
$ curl localhost:9404  
# HELP jmx_config_reload_failure_total Number of times configuration have failed to be  
reloaded.  
# TYPE jmx_config_reload_failure_total counter
```

```
jmx_config_reload_failure_total 0.0
```

2. Create the Prometheus service discovery YAML file. The following example service discovery file performs the following:

- Specifies the Prometheus JMX exporter host port as `localhost: 9404`.
- Attaches labels (`Application`, `ComponentName`, and `InstanceId`) to the metrics, which can be set as CloudWatch metric dimensions.

```
$ cat prometheus_sd_jmx.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    Application: myApp
    ComponentName: arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/
app/saml-Appli-MMZW8E3GH4H2/aac36d7fea2a6e5b
    InstanceId: i-12345678901234567
```

3. Create the Prometheus JMX exporter configuration YAML file. The following example configuration file specifies the following:

- The metrics retrieval job interval and timeout period.
- The metrics retrieval jobs (`jmx` and `sap`), also known as scraping, which include the job name, maximum time series returned at a time, and service discovery file path.

```
$ cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_jmx.yaml"]
  - job_name: sap
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_sap.yaml"]
```

4. Verify that the CloudWatch agent is installed on your Amazon EC2 instance and that the version is 1.247346.1b249759 or later. To install the CloudWatch agent on your EC2 instance, see [Installing the CloudWatch Agent](#). To verify the version, see [Finding information about CloudWatch agent versions](#).
5. Configure the CloudWatch agent. For more information about how to configure the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file](#). The following example CloudWatch agent configuration file performs the following:
 - Specifies the Prometheus JMX exporter configuration file path.
 - Specifies the target log group to which to publish EMF metric logs.
 - Specifies two sets of dimensions for each metric name.
 - Sends 8 (4 metric names * 2 sets of dimensions per metric name) CloudWatch metrics.

```
{
  "logs":{
    "logs_collected":{
      ....
```

```

},
"metrics_collected": {
    "prometheus": {
        "cluster_name": "prometheus-test-cluster",
        "log_group_name": "prometheus-test",
        "prometheus_config_path": "/tmp/prometheus.yaml",
        "emf_processor": {
            "metric_declaration_dedup": true,
            "metric_namespace": "CWAgent",
            "metric_unit": {
                "jvm_threads_current": "Count",
                "jvm_gc_collection_seconds_sum": "Second",
                "jvm_memory_bytes_used": "Bytes"
            },
            "metric_declaration": [
                {
                    "source_labels": [
                        "job"
                    ],
                    "label_matcher": "^jmx$",
                    "dimensions": [
                        [
                            "InstanceId",
                            "ComponentName"
                        ],
                        [
                            "ComponentName"
                        ]
                    ],
                    "metric_selectors": [
                        "^java_lang_threading_threadcount$",
                        "^java_lang_memory_heapmemoryusage_used$",
                        "^java_lang_memory_heapmemoryusage_committed$"
                    ]
                }
            ]
        }
    },
    "metrics": {
        ...
    }
}

```

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure

You can retrieve workload-specific SAP HANA metrics from [Prometheus HANA database exporter](#) for Application Insights to configure and monitor alarms. For more information, see [Set up your SAP HANA database for monitoring \(p. 679\)](#) in this guide.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the AWS-provided Prometheus HANA database exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HANA database exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP HANA database
- Linux operating system (SUSE Linux, RedHat Linux)

- A secret with SAP HANA database monitoring credentials, using AWS Secrets Manager. Create a secret using the key/value pairs format, specify the key username, and enter the database user for the value. Add a second key password, and then enter the password for the value. For more information about how to create secrets, see [Create a secret](#) in the *AWS Secrets Manager User Guide*. The secret must be formatted as follows:

```
{
  "username": "<database_user>",
  "password": "<database_password>"
}
```

Install and configure the AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure package

The `AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure [Prometheus HANA database Exporter](#). When HANA database metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

- Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter configurations. The following is an example parameter value.

```
{"\exposition_port":9668,"multi_tenant":true,"timeout":600,"hana":{\"host\": \"localhost\", \"port\":30013,\"aws_secret_name\":\"HANA_DB_CREDS\", \"scale_out_mode\":true}}
```

Note

In this example, the export runs only on the Amazon EC2 instance with the active `SYSTEM` database, and it will remain idle on the other EC2 instances in order to avoid duplicate metrics. The exporter can retrieve all of the database tenant information from the `SYSTEM` database.

- Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter metrics queries. The package can accept more than one metrics parameter. Each parameter must have a valid JSON object format. The following is an example parameter value:

```
{"SELECT MAX(TIMESTAMP) TIMESTAMP, HOST, MEASURED_ELEMENT_NAME CORE, SUM(MAP(CAPTION, 'User Time', TO_NUMBER(VALUE), 0)) USER_PCT, SUM(MAP(CAPTION, 'System Time', TO_NUMBER(VALUE), 0)) SYSTEM_PCT, SUM(MAP(CAPTION, 'Wait Time', TO_NUMBER(VALUE), 0)) WAITIO_PCT, SUM(MAP(CAPTION, 'Idle Time', 0, TO_NUMBER(VALUE))) BUSY_PCT, SUM(MAP(CAPTION, 'Idle Time', TO_NUMBER(VALUE), 0)) IDLE_PCT FROM sys.M_HOST_AGENT_METRICS WHERE MEASURED_ELEMENT_TYPE = 'Processor' GROUP BY HOST, MEASURED_ELEMENT_NAME;": [{"name": "hanadb_cpu_user", "description": "Percentage of CPU time spent by HANA DB in user space, over the last minute (in seconds)", "labels": [{"HOST": "", "CORE": ""}], "value": "USER_PCT", "unit": "%", "type": "gauge"}, {"name": "hanadb_cpu_system", "description": "Percentage of CPU time spent by HANA DB in Kernel space, over the last minute (in seconds)", "labels": [{"HOST": "", "CORE": ""}], "value": "SYSTEM_PCT", "unit": "%", "type": "gauge"}, {"name": "hanadb_cpu_waitio", "description": "Percentage of CPU time spent by HANA DB in IO mode, over the last minute (in seconds)", "labels": [{"HOST": "", "CORE": ""}], "value": "WAITIO_PCT", "unit": "%", "type": "gauge"}, {"name": "hanadb_cpu_busy", "description": "Percentage of CPU time spent by HANA DB, over the last minute (in seconds)", "labels": [{"HOST": "", "CORE": ""}], "value": "BUSY_PCT", "unit": "%", "type": "gauge"}, {"name": "hanadb_cpu_idle", "description": "Percentage of CPU time not spent by HANA DB, over the last minute (in seconds)", "labels": [{"HOST": "", "CORE": ""}], "value": "IDLE_PCT", "unit": "%", "type": "gauge"}]}
```

For more information about metrics queries, see the [SUSE / hanadb_exporter](#) repo on GitHub.

3. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure*** and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
    "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}",
    "SSM_SID": "<SAP_DATABASE_SID>",
    "SSM_EXPORTER_METRICS_1": "{{ssm:<SSM_FIRST_METRICS_PARAMETER_STORE_NAME>}}",
    "SSM_EXPORTER_METRICS_2": "{{ssm:<SSM_SECOND_METRICS_PARAMETER_STORE_NAME>}}"
}
```

5. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure

You can retrieve workload-specific High Availability (HA) cluster metrics from [Prometheus HANA cluster exporter](#) for Application Insights to configure and monitor alarms for an SAP HANA database High Availability setup. For more information, see [Set up your SAP HANA database for monitoring \(p. 679\)](#) in this guide.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the AWS-provided Prometheus HA cluster exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HA cluster exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- HA cluster for Pacemaker, Corosync, SBD, and DRBD
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the AWSObservabilityExporter-HAClusterExporterInstallAndConfigure package

The **AWSObservabilityExporter-HAClusterExporterInstallAndConfigure** package is an SSM Distributor package that you can use to install and configure Prometheus HA Cluster Exporter. When cluster metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter configurations in JSON format. The following is an example parameter value.

```
{"port": "9664", "address": "0.0.0.0", "log-level": "info", "crm-mon-path": "/usr/sbin/crm_mon", "cibadmin-path": "/usr/sbin/cibadmin", "corosync-cfgtoolpath-path": "/usr/sbin/corosync-cfgtool", "corosync-quorumtool-path": "/usr/sbin/corosync-quorumtool", "sbd-path": "/usr/sbin/sbd", "sbd-config-path": "/etc/sysconfig/sbd", "drbdsetup-path": "/sbin/drbdsetup", "enable-timestamps": false}
```

For more information about the exporter configurations, see the [ClusterLabs / ha_cluster_exporter](#) repo on GitHub.

2. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-HAClusterExporterInstallAndConfigure*** and choose **Install one time**.

3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{  
    "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}"
```

4. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

Get started with Amazon CloudWatch Application Insights

To get started with CloudWatch Application Insights, verify that you have met the following prerequisites and have created an IAM policy. Then, you can get started using the console link to enable CloudWatch Application Insights. To configure your application resources, follow the steps under [Set up, configure, and manage your application for monitoring \(p. 617\)](#).

Contents

- [Access CloudWatch Application Insights \(p. 615\)](#)
- [Prerequisites \(p. 615\)](#)
- [IAM policy \(p. 616\)](#)
- [IAM role permissions for account-based application onboarding \(p. 617\)](#)
- [Set up, configure, and manage your application for monitoring \(p. 617\)](#)

Access CloudWatch Application Insights

You can access and manage CloudWatch Application Insights through one of the following interfaces:

- **CloudWatch console.** To add monitors for your application, choose **Application Insights** under **Insights** in the left navigation pane of the [CloudWatch console](#). After your application is configured, you can use the [CloudWatch console](#) to view and analyze problems that are detected.
- **AWS Command Line Interface (AWS CLI).** You can use the AWS CLI to access AWS API operations. For more information, see [Installing the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#). For **Application Insights** API information, see the [Amazon CloudWatch Application Insights API Reference](#).

Prerequisites

You must complete the following prerequisites to configure an application with CloudWatch Application Insights:

- **AWS SSM enablement.** You must install Systems Manager Agent (SSM Agent), and your instances must be enabled for SSM. For information about how to install the SSM Agent, see [Setting Up AWS SSM](#).
- **EC2 instance role.** You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager (see [Using Identity-based Policies \(IAM Policies\) for AWS SSM](#)) and the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. See [Create IAM Roles and Users for Use With CloudWatch Agent](#) for more information.
- **AWS resource groups.** To onboard your applications to CloudWatch Application Insights, you must create a resource group that includes all associated AWS resources used by your application stack. This

includes application load balancers, EC2 instances running IIS and web front-end, .NET worker tiers, and your SQL Server database. CloudWatch Application Insights automatically includes Auto Scaling groups using the same tags or CloudFormation stacks as your resource group, because Auto Scaling groups are not currently supported by resource groups. For more information, see [Getting Started with AWS Resource Groups](#).

- **IAM permissions:** For non-admin users, you must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps on attaching the policy, see [IAM policy \(p. 616\)](#).
- **Service-linked role:** CloudWatch Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first CloudWatch Application Insights application in the AWS Management Console. For more information, see [Using service-linked roles for CloudWatch Application Insights \(p. 930\)](#).
- **Performance Counter metrics support for EC2 Windows instances:** To monitor Performance Counter metrics on your EC2 Windows instances, Performance Counters must be installed on the instances. For Performance Counter metrics and corresponding Performance Counter set names, see [Performance Counter metrics \(p. 740\)](#). For more information about Performance Counters, see [Performance Counters](#).

IAM policy

To use CloudWatch Application Insights, you must create an [Identity and Access Management \(IAM\) policy](#) and attach it to your IAM user identity. The IAM policy defines the user permissions.

To create an IAM policy using the console

To create an IAM policy using the IAM console, perform the following steps.

1. Go to the [IAM console](#). In the left navigation pane, select **Policies**.
2. At the top of the page, select **Create policy**.
3. Select the **JSON** tab.
4. Copy and paste the following JSON document under the **JSON** tab.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "applicationinsights:*",  
                "iam:CreateServiceLinkedRole",  
                "iam>ListRoles",  
                "resource-groups>ListGroups"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

5. Select **Review Policy**.
6. Enter a **Name** for the policy, for example, “ApplInsightsPolicy.” Optionally, enter a **Description**.
7. Select **Create Policy**.
8. Select **Users** from the left navigation pane.
9. Select the **User name** of the user to which you would like to attach the policy.
10. Select **Add permissions**.
11. Select **Attach existing policies directly**.

12. Search for the policy that you just created, and select the check box to the left of the policy name.
13. Select **Next: Review**.
14. Make sure that the correct policy is listed, and select **Add permissions**.
15. Make sure that you log in with the user associated with the policy that you just created when you use CloudWatch Application Insights.

To create an IAM policy using the AWS CLI

To create an IAM policy using the AWS CLI, run the [create-policy](#) operation from the command line using the JSON document above as a file in your current folder.

To create an IAM policy using AWS Tools for Windows PowerShell

To create an IAM policy using the AWS Tools for Windows PowerShell, run the [New-IAMPolicy](#) cmdlet using the JSON document above as a file in your current folder.

IAM role permissions for account-based application onboarding

If you want to onboard all of the resources in your account, and you choose not to use the [Application Insights managed policy \(p. 937\)](#) for full access to Application Insights functionality, you must attach the following permissions to your IAM role so that Application Insights can discover all of the resources in your account:

```
"ec2:DescribeInstances"
"ec2:DescribeVolumes"
"rds:DescribeDBInstances"
"rds:DescribeDBClusters"
"sqs:ListQueues"
"elasticloadbalancing:DescribeLoadBalancers"
"autoscaling:DescribeAutoScalingGroups"
"lambda>ListFunctions"
"dynamodb>ListTables"
"s3>ListAllMyBuckets"
"sns>ListTopics"
"states>ListStateMachines"
"apigateway:GET"
"ecs>ListClusters"
"ecs:DescribeTaskDefinition"
"ecs>ListServices"
"ecs>ListTasks"
"eks>ListClusters"
"eks>ListNodegroups"
"fsx:DescribeFileSystems"
```

Set up, configure, and manage your application for monitoring

This section provides steps to set up, configure, and manage your CloudWatch Application Insights application using the console, the AWS CLI, and AWS Tools for Windows PowerShell.

Topics

- [Set up, configure, and manage your application for monitoring from the CloudWatch console \(p. 618\)](#)
- [Set up, configure, and manage your application for monitoring using the command line \(p. 621\)](#)

- [Application Insights CloudWatch Events and notifications for detected problems \(p. 633\)](#)

Set up, configure, and manage your application for monitoring from the CloudWatch console

This section provides steps to set up, configure, and manage your application for monitoring from the CloudWatch console.

Console procedures

- [Add and configure an application \(p. 618\)](#)
- [Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring \(p. 620\)](#)
- [Disable monitoring for an application component \(p. 621\)](#)
- [Delete an application \(p. 621\)](#)

Add and configure an application

Add and configure an application from the CloudWatch console

To get started with CloudWatch Application Insights from the CloudWatch console, perform the following steps.

1. **Start.** Open the [CloudWatch console landing page](#). From the left navigation pane, under **Insights**, choose **Application Insights**. The page that opens shows the list of applications that are monitored with CloudWatch Application Insights, along with their monitoring status.
2. **Add an application.** To set up monitoring for your application, choose **Add an application**. When you choose **Add an application**, you are prompted to **Choose Application Type**.
 - **Resource group-based application.** When you select this option, you can choose which resource groups in this account to monitor.
 - **Account-based application.** When you select this option, you can monitor all of the resources in this account. If you want to monitor all of the resources in an account, we recommend this option over the resource group-based option because the application onboarding process is faster.

Note

You cannot combine resource group-based monitoring with account-based monitoring using Application Insights. In order to change the application type, you must delete all of the applications that are being monitored, and **Choose Application Type**.

When you add your first application for monitoring, CloudWatch Application Insights creates a service-linked role in your account, which gives Application Insights permissions to call other AWS services on your behalf. For more information about the service-linked role created in your account by Application Insights, see [Using service-linked roles for CloudWatch Application Insights \(p. 930\)](#).

3. Resource-based application monitoring
 1. **Select resource group.** On the **Specify application details** page, select the AWS resource group that contains your application resources from the dropdown list. These resources include front-end servers, load balancers, auto scaling groups, and database servers.

If you have not created a resource group for your application, you can create one by choosing **Create new resource group**. For more information about creating resource groups, see the [AWS Resource Groups User Guide](#).
 2. **Monitor CloudWatch Events.** Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS

CodeDeploy, Amazon ECS, AWS Health APIs And Notifications, Amazon RDS, Amazon S3, and AWS Step Functions.

3. **Integrate with AWS Systems Manager OpsCenter.** To view and get notified when problems are detected for selected applications, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your AWS resources, provide the SNS topic ARN.
4. **Tags — optional.** CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see [Working with Tag Editor](#).
5. Choose **Next**.

An [ARN](#) is generated for the application in the following format.

```
arn:partition:applicationinsights:region:account-id:application/resource-group/resource-group-name
```

Example

```
arn:aws:applicationinsights:us-east-1:123456789012:application/resource-group/my-resource-group
```

Account-based application monitoring

1. **Application name.** Enter a name for your account-based application.
2. **Automated monitoring of new resources.** By default, Application Insights uses recommended settings to configure monitoring for resource components that are added to your account after you onboard the application. You can exclude monitoring for resources added after onboarding your application by clearing the check box.
3. **Monitor CloudWatch Events.** Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs And Notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
4. **Integrate with AWS Systems Manager OpsCenter.** To view and get notified when problems are detected for selected applications, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your AWS resources, provide the SNS topic ARN.
5. **Tags — optional.** CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see [Working with Tag Editor](#).
6. **Discovered resources.** All of the resources discovered in your account are added to this list. If Application Insights is unable to discover all of the resources in your account, an error message appears at the top of the page. This message includes a link to the [documentation for how to add the required permissions \(p. 617\)](#).
7. Choose **Next**.

An [ARN](#) is generated for the application in the following format.

```
arn:partition:applicationinsights:region:account-id:application/TBD/application-name
```

Example

```
arn:aws:applicationinsights:us-east-1:123456789012:application/TBD/my-application
```

4. After you submit your application monitoring configuration, you will be taken to the details page for the application, where you can view the **Application summary**, the list of **Monitored components** and **Unmonitored components**, and, by selecting the tabs next to **Components**, the **Configuration history**, **Log patterns**, and any **Tags** that you have applied.

To view insights for the application, choose **View Insights**.

You can update your selections for CloudWatch Events monitoring and integration with AWS Systems Manager OpsCenter by choosing **Edit**.

Under **Components**, you can select the **Actions** menu to Create, Modify, or Ungroup an instance group.

You can manage monitoring for components, including application tier, log groups, event logs, metrics, and custom alarms, by selecting the bullet next to a component and choosing **Manage monitoring**.

Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring

You can enable Application Insights to monitor containerized applications and microservices from the Container Insights console. Application Insights supports monitoring for the following resources:

- Amazon ECS clusters
- Amazon ECS services
- Amazon ECS tasks
- Amazon EKS clusters

When Application Insights is enabled, it provides recommended metrics and logs, detects potential problems, generates CloudWatch Events, and creates automatic dashboards for your containerized applications and microservices.

You can enable Application Insights for containerized resources from the Container Insights or Application Insights consoles.

Enable Application Insights from the Container Insights console

From the Container Insights console, on the Container Insights **Performance monitoring** dashboard, choose **Auto-configure Application Insights**. When Application Insights is enabled, it displays details about detected problems.

Enable Application Insights from the Application Insights console

When ECS clusters appear in the component list, Application Insights automatically enables additional container monitoring with Container Insights.

For EKS clusters, you can enable additional monitoring with Container Insights to provide diagnostics information, such as container restart failures, to help you isolate and resolve problems. Additional steps are required to set up Container Insights for EKS. For information, see [Setting up Container Insights on Amazon EKS and Kubernetes \(p. 320\)](#) for steps to set up Container Insights on EKS.

Additional monitoring for EKS with Container Insights is supported on Linux instances with EKS.

For more information about Container Insights support for ECS and EKS clusters, see [Using Container Insights \(p. 305\)](#).

Disable monitoring for an application component

To disable monitoring for an application component, from the application details page, select the component for which you want to disable monitoring. Choose **Actions**, and then **Remove from monitoring**.

Delete an application

To delete an application, from the CloudWatch dashboard, on the left navigation pane, choose **Application Insights** under **Insights**. Select the application that you want to delete. Under **Actions**, choose **Delete application**. This deletes monitoring and deletes all of the saved monitors for application components. The application resources are not deleted.

Set up, configure, and manage your application for monitoring using the command line

This section provides steps for setting up, configuring, and managing your application for monitoring using the AWS CLI and AAWS Tools for Windows PowerShell.

Command line procedures

- [Add and manage an application \(p. 621\)](#)
- [Manage and update monitoring \(p. 624\)](#)
- [Configure monitoring for SQL Always On Availability Groups \(p. 627\)](#)
- [Configure monitoring for MySQL RDS \(p. 630\)](#)
- [Configure monitoring for MySQL EC2 \(p. 630\)](#)
- [Configure monitoring for PostgreSQL RDS \(p. 631\)](#)
- [Configure monitoring for PostgreSQL EC2 \(p. 631\)](#)
- [Configure monitoring for Oracle RDS \(p. 632\)](#)
- [Configure monitoring for Oracle EC2 \(p. 632\)](#)

Add and manage an application

You can add, get information about, manage, and configure your Application Insights application using the command line.

Topics

- [Add an application \(p. 621\)](#)
- [Describe an application \(p. 622\)](#)
- [List components in an application \(p. 622\)](#)
- [Describe a component \(p. 622\)](#)
- [Group similar resources into a custom component \(p. 623\)](#)
- [Ungroup a custom component \(p. 623\)](#)
- [Update an application \(p. 624\)](#)
- [Update a custom component \(p. 624\)](#)

Add an application

Add an application using the AWS CLI

To use the AWS CLI to add an application for your resource group called `my-resource-group`, with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`, use the following command.

```
aws application-insights create-application --resource-group-name my-resource-group --ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Add an application using AWS Tools for Windows PowerShell

To use AWS Tools for Windows PowerShell to add an application for your resource group called `my-resource-group` with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`, use the following command.

```
New-CWAIAApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Describe an application

Describe an application using the AWS CLI

To use the AWS CLI to describe an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights describe-application --resource-group-name my-resource-group
```

Describe an application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe an application created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIAApplication -ResourceGroupName my-resource-group
```

List components in an application

List components in an application using the AWS CLI

To use the AWS CLI to list the components created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-components --resource-group-name my-resource-group
```

List components in an application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to list the components created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIComponentList -ResourceGroupName my-resource-group
```

Describe a component

Describe a component using the AWS CLI

You can use the following AWS CLI command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
aws application-insights describe-component --resource-group-name my-resource-group --component-name my-component
```

Describe a component using AWS Tools for Windows PowerShell

You can use the following AWS Tools for Windows PowerShell command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
Get-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Group similar resources into a custom component

We recommend grouping similar resources, such as .NET web server instances, into custom components for easier onboarding and better monitoring and insights. Currently, CloudWatch Application Insights supports custom groups for EC2 instances.

To group resources into a custom component using the AWS CLI

To use the AWS CLI to group three instances (`arn:aws:ec2:us-east-1:123456789012:instance/i-11111`, `arn:aws:ec2:us-east-1:123456789012:instance/i-22222`, and `arn:aws:ec2:us-east-1:123456789012:instance/i-33333`) together into a custom component called `my-component` for an application created for the resource group called `my-resource-group`, use the following command.

```
aws application-insights create-component --resource-group-name my-resource-group --component-name my-component --resource-list arn:aws:ec2:us-east-1:123456789012:instance/i-11111 arn:aws:ec2:us-east-1:123456789012:instance/i-22222 arn:aws:ec2:us-east-1:123456789012:instance/i-33333
```

To group resources into a custom component using AWS Tools for Windows PowerShell

To use AWS Tools for Windows PowerShell to group three instances (`arn:aws:ec2:us-east-1:123456789012:instance/i-11111`, `arn:aws:ec2:us-east-1:123456789012:instance/i-22222`, and `arn:aws:ec2:us-east-1:123456789012:instance/i-33333`) together into a custom component called `my-component`, for an application created for the resource group called `my-resource-group`, use the following command.

```
New-CWAIComponent -ResourceGroupName my-resource-group -ComponentName my-component -ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-11111,arn:aws:ec2:us-east-1:123456789012:instance/i-22222,arn:aws:ec2:us-east-1:123456789012:instance/i-33333
```

Ungroup a custom component

To ungroup a custom component using the AWS CLI

To use the AWS CLI to ungroup a custom component named `my-component` in an application created on the resource group, `my-resource-group`, use the following command.

```
aws application-insights delete-component --resource-group-name my-resource-group --component-name my-new-component
```

To ungroup a custom component using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to ungroup a custom component named `my-component` in an application created on the resource group, `my-resource-group`, use the following command.

```
Remove-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Update an application

Update an application using the AWS CLI

You can use the AWS CLI to update an application to generate AWS Systems Manager OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws:sns:us-east-1:123456789012:MyTopic`, using the following command.

```
aws application-insights update-application --resource-group-name my-resource-group --ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Update an application using AWS Tools for Windows PowerShell

You can use the AWS Tools for Windows PowerShell to update an application to generate AWS SSM OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws:sns:us-east-1:123456789012:MyTopic`, using the following command.

```
Update-CWAIAApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Update a custom component

Update a custom component using the AWS CLI AWS CLI

You can use the AWS CLI to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
aws application-insights update-component --resource-group-name my-resource-group --component-name my-component --new-component-name my-new-component --resource-list arn:aws:ec2:us-east-1:123456789012:instance/i-44444 arn:aws:ec2:us-east-1:123456789012:instance/i-55555
```

Update a custom component using AWS Tools for Windows PowerShell

You can use the AWS Tools for Windows PowerShell to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
Update-CWAIComponent -ComponentName my-component -NewComponentName my-new-component -ResourceGroupName my-resource-group -ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-44444,arn:aws:ec2:us-east-1:123456789012:instance/i-55555
```

Manage and update monitoring

You can manage and update monitoring for your Application Insights application using the command line.

Topics

- [List problems with your application \(p. 625\)](#)
- [Describe an application problem \(p. 625\)](#)
- [Describe the anomalies or errors associated with a problem \(p. 625\)](#)
- [Describe an anomaly or error with the application \(p. 625\)](#)
- [Describe the monitoring configurations of a component \(p. 626\)](#)
- [Describe the recommended monitoring configuration of a component \(p. 626\)](#)
- [Update the monitoring configurations for a component \(p. 627\)](#)
- [Remove a specified resource group from Application Insights monitoring \(p. 627\)](#)

List problems with your application

List problems with your application using the AWS CLI

To use the AWS CLI to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-problems --resource-group-name my-resource-group --start-time 1000 --end-time 10000
```

List problems with your application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
$startDate = "8/6/2019 3:33:00"  
$endDate = "8/6/2019 3:34:00"  
Get-CWAIPProblemList -ResourceGroupName my-resource-group -StartTime $startDate -  
EndTime $endDate
```

Describe an application problem

Describe an application problem using the AWS CLI

To use the AWS CLI to describe a problem with problem id `p-1234567890`, use the following command.

```
aws application-insights describe-problem --problem-id p-1234567890
```

Describe an application problem using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe a problem with problem id `p-1234567890`, use the following command.

```
Get-CWAIPProblem -ProblemId p-1234567890
```

Describe the anomalies or errors associated with a problem

Describe the anomalies or errors associated with a problem using the AWS CLI

To use the AWS CLI to describe the anomalies or errors associated with a problem with problem id `p-1234567890`, use the following command.

```
aws application-insights describe-problem-observations --problem-id -1234567890
```

Describe the anomalies or errors associated with a problem using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe the anomalies or errors associated with a problem with problem id `p-1234567890`, use the following command.

```
Get-CWAIPProblemObservation -ProblemId p-1234567890
```

Describe an anomaly or error with the application

Describe an anomaly or error with the application using the AWS CLI

To use the AWS CLI to describe an anomaly or error with the application with the observation id o-1234567890, use the following command.

```
aws application-insights describe-observation --observation-id o-1234567890
```

Describe an anomaly or error with the application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe an anomaly or error with the application with the observation id o-1234567890, use the following command.

```
Get-CWAIObservation -ObservationId o-1234567890
```

Describe the monitoring configurations of a component

Describe the monitoring configurations of a component using the AWS CLI

To use the AWS CLI to describe the monitoring configuration of a component called my-component in an application created on the resource group my-resource-group, use the following command.

```
aws application-insights describe-component-configuration --resource-group-name my-resource-group --component-name my-component
```

Describe the monitoring configurations of a component using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe the monitoring configuration of a component called my-component, in an application created on the resource group my-resource-group, use the following command.

```
Get-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-resource-group
```

For more information about component configuration and for example JSON files, see [Work with component configurations \(p. 634\)](#).

Describe the recommended monitoring configuration of a component

Describe the recommended monitoring configuration of a component using the AWS CLI

When the component is part of a .NET Worker application, you can use the AWS CLI to describe the recommended monitoring configuration of a component called my-component in an application created on the resource group my-resource-group, by using the following command.

```
aws application-insights describe-component-configuration-recommendation --resource-group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER
```

Describe the recommended monitoring configuration of a component using AWS Tools for Windows PowerShell

When the component is part of a .NET Worker application, you can use the AWS Tools for Windows PowerShell to describe the recommended monitoring configuration of a component called my-component in an application created on the resource group my-resource-group, by using the following command.

```
Get-CWAIComponentConfigurationRecommendation -ComponentName my-component -ResourceGroupName my-resource-group -Tier DOT_NET_WORKER
```

For more information about component configuration and for example JSON files, see [Work with component configurations \(p. 634\)](#).

Update the monitoring configurations for a component

Update the monitoring configurations for a component using the AWS CLI

To use the AWS CLI to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
aws application-insights update-component-configuration --resource-group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER --monitor --component-configuration "file://configuration.txt"
```

Update the monitoring configurations for a component using the AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
[string]$config = Get-Content -Path configuration.txt
Update-CWAICComponentConfiguration -ComponentName my-component -ResourceGroupName my-resource-group -Tier DOT_NET_WORKER -Monitor 1 -ComponentConfiguration $config
```

For more information about component configuration and for example JSON files, see [Work with component configurations \(p. 634\)](#).

Remove a specified resource group from Application Insights monitoring

Remove a specified resource group from Application Insights monitoring using the AWS CLI

To use the AWS CLI to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
aws application-insights delete-application --resource-group-name my-resource-group
```

Remove a specified resource group from Application Insights monitoring using the AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
Remove-CWAIAApplication -ResourceGroupName my-resource-group
```

Configure monitoring for SQL Always On Availability Groups

1. Create an application for the resource group with the SQL HA EC2 instances.

```
aws application-insights create-application #-region <REGION> #-resource-group-name  
<RESOURCE_GROUP_NAME>
```

2. Define the EC2 instances that represent the SQL HA cluster by creating a new application component.

```
aws application-insights create-component #-resource-group-name  
<RESOURCE_GROUP_NAME> #-component-name SQL_HA_CLUSTER #-resource-list  
"arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_1_ID>"  
"arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_2_ID>"
```

3. Configure the SQL HA component.

```
aws application-insights update-component-configuration #-resource-group-name  
<RESOURCE_GROUP_NAME> #-region <REGION> #-component-name "SQL_HA_CLUSTER" #-monitor #-tier  
SQL_SERVER_ALWAYS_ON_AVAILABILITY_GROUP #-monitor #-component-configuration '{  
    "subComponents" : [ {  
        "subComponentType" : "AWS::EC2::Instance",  
        "alarmMetrics" : [ {  
            "alarmMetricName" : "CPUUtilization",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "StatusCheckFailed",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Processor % Processor Time",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Memory % Committed Bytes In Use",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Memory Available Mbytes",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Paging File % Usage",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "System Processor Queue Length",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Network Interface Bytes Total/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "PhysicalDisk % Disk Time",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "SQLServer:General Statistics Processes blocked",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "SQLServer:General Statistics User Connections",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",  
            "monitor" : true  
        }, {
```

```

    "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
    "monitor" : true
  }, {
    "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
    "monitor" : true
  } ],
  "windowsEvents" : [ {
    "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
    "eventName" : "Application",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
    "monitor" : true
  }, {
    "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
    "eventName" : "System",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
  }, {
    "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
    "eventName" : "Security",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
  } ],
  "logs" : [ {
    "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
    "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\\MSSQL\\
Log\\ERRORLOG",
    "logType" : "SQL_SERVER",
    "monitor" : true,
    "encoding" : "utf-8"
  } ]
}, {
  "subComponentType" : "AWS::EC2::Volume",
  "alarmMetrics" : [ {
    "alarmMetricName" : "VolumeReadBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeReadOps",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteOps",
    "monitor" : true
  }
]
}

```

```

        "alarmMetricName" : "VolumeWriteOps",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeQueueLength",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeThroughputPercentage",
        "monitor" : true
    }, {
        "alarmMetricName" : "BurstBalance",
        "monitor" : true
    } ]
}
}'

```

Note

Application Insights must ingest Application Event logs (information level) to detect cluster activities such as failover.

Configure monitoring for MySQL RDS

1. Create an application for the resource group with the RDS MySQL database instance.

```
aws application-insights create-application #--region <REGION> #--resource-group-name
<RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see [Accessing the MySQL Slow Query and General Logs](#).
 - set slow_query_log = 1
 - set log_output = FILE
3. Export the logs to be monitored to CloudWatch logs. For more information, see [Publishing MySQL Logs to CloudWatch Logs](#).
4. Configure the MySQL RDS component.

```
aws application-insights update-component-configuration #--resource-group-name
<RESOURCE_GROUP_NAME> #--region <REGION> #--component-name "<DB_COMPONENT_NAME>" #--monitor #--tier DEFAULT #--monitor #--component-configuration "{\"alarmMetrics\": [{\"alarmMetricName\":\"CPUUtilization\", \"monitor\":true}], \"logs\": [{\"logType\": \"MYSQL\", \"monitor\":true}, {\"logType\": \"MYSQL_SLOW_QUERY\", \"monitor\":false}]}"
```

Configure monitoring for MySQL EC2

1. Create an application for the resource group with the SQL HA EC2 instances.

```
aws application-insights create-application #--region <REGION> #--resource-group-name
<RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see [Accessing the MySQL Slow Query and General Logs](#).
 - set slow_query_log = 1
 - set log_output = FILE
3. Configure the MySQL EC2 component.

```
aws application-insights update-component-configuration #--resource-group-name
<RESOURCE_GROUP_NAME> #--region <REGION> #--component-name "<DB_COMPONENT_NAME>"
```

```
#-monitor #-tier MYSQL #-monitor #	component-configuration "{\"alarmMetrics\": [{\"alarmMetricName\":\"CPUUtilization\", \"monitor\":true}], \"logs\":[{\"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\", \"logPath\":\"C:\\\\ProgramData\\\\MySQL\\\\MySQL Server *\\\\Data\\\\<FILE_NAME>.err\", \"logType\":\"MYSQL\", \"monitor\":true, \"encoding\":\"utf-8\"}]}"]}
```

Configure monitoring for PostgreSQL RDS

1. Create an application for the resource group with the PostgreSQL RDS database instance.

```
aws application-insights create-application #-region <REGION> #-resource-group-name <RESOURCE_GROUP_NAME>
```

2. Publishing PostgreSQL logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check box labeled **PostgreSQL log**. Choose **Continue** to save this setting.
3. Your PostgreSQL logs are exported to CloudWatch.
4. Configure the PostgreSQL RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --tier DEFAULT --component-configuration
"{
    \"alarmMetrics\": [
        {
            \"alarmMetricName\": \"CPUUtilization\",
            \"monitor\": true
        }
    ],
    \"logs\": [
        {
            \"logType\": \"POSTGRESQL\",
            \"monitor\": true
        }
    ]
}"
```

Configure monitoring for PostgreSQL EC2

1. Create an application for the resource group with the PostgreSQL EC2 instance.

```
aws application-insights create-application #-region <REGION> #-resource-group-name <RESOURCE_GROUP_NAME>
```

2. Configure the PostgreSQL EC2 component.

```
aws application-insights update-component-configuration #-region <REGION> #-resource-group-name <RESOURCE_GROUP_NAME> #-component-name <DB_COMPONENT_NAME> #-monitor #-tier POSTGRESOL #-component-configuration
"{
    \"alarmMetrics\": [
        {
            \"alarmMetricName\": \"CPUUtilization\",
            \"monitor\":true
        }
    ],
    \"logs\": [
        {
            \"logType\": \"POSTGRESOL\",
            \"monitor\": true
        }
    ]
}"
```

```

        \\"logGroupName\":\\"<UNIQUE_LOG_GROUP_NAME>\",
        \\"logPath\":\\"/var/lib/pgsql/data/log\\",
        \\"logType\":\"POSTGRESQL\",
        \\"monitor\":true,
        \\"encoding\":\"utf-8\"
    }
]
}"

```

Configure monitoring for Oracle RDS

1. Create an application for the resource group with the Oracle RDS database instance.

```
aws application-insights create-application #‐‐region <REGION> #‐‐resource-group-name
<RESOURCE_GROUP_NAME>
```

2. Publishing Oracle logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check boxes labeled **Alert log** and **Listener log**. Choose **Continue** to save this setting.
3. Your Oracle logs are exported to CloudWatch.
4. Configure the Oracle RDS component.

```

aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --tier
DEFAULT --component-configuration
"{
    \\"alarmMetrics\":[
        {
            \\"alarmMetricName\": \"CPUUtilization\",
            \\"monitor\": true
        }
    ],
    \\"logs\":[
        {
            \\"logType\": \"ORACLE_ALERT\",
            \\"monitor\": true
        },
        {
            \\"logType\": \"ORACLE_LISTENER\",
            \\"monitor\": true
        }
    ]
}"

```

Configure monitoring for Oracle EC2

1. Create an application for the resource group with the Oracle EC2 instance.

```
aws application-insights create-application #‐‐region <REGION> #‐‐resource-group-name
<RESOURCE_GROUP_NAME>
```

2. Configure the Oracle EC2 component.

```

aws application-insights update-component-configuration #‐‐region <REGION> #‐‐resource-
group-name <RESOURCE_GROUP_NAME> #‐‐component-name <DB_COMPONENT_NAME> #‐‐monitor #‐‐tier
ORACLE #‐‐component-configuration
"{
    \\"alarmMetrics\":[

```

```
{  
    \\"alarmMetricName\\":\\"CPUUtilization\\",  
    \\"monitor\\":true  
}  
,  
\\"logs\\": [  
    {  
        \\"logGroupName\\":\\"<UNIQUE_LOG_GROUP_NAME>\",  
        \\"logPath\\": \"/opt/oracle/diag/rdbms/*/*/trace\",  
        \\"logType\\":\\"ORACLE_ALERT\\",  
        \\"monitor\\":true,  
    },  
    {  
        \\"logGroupName\\":\\"<UNIQUE_LOG_GROUP_NAME>\",  
        \\"logPath\\": \"/opt/oracle/diag/tnslsnr/$HOSTNAME/listener/trace/\",  
        \\"logType\\":\\"ORACLE_ALERT\\",  
        \\"monitor\\":true,  
    }  
]  
}"
```

Application Insights CloudWatch Events and notifications for detected problems

For each application that is added to CloudWatch Application Insights, a CloudWatch event is published for the following events on a best effort basis:

- **Problem creation.** Emitted when CloudWatch Application Insights detects a new problem.
 - Detail Type: **"Application Insights Problem Detected"**
 - Detail:
 - **problemId:** The detected problem ID.
 - **region:** The AWS Region where the problem was created.
 - **resourceGroupName:** The Resource Group for the registered application for which the problem was detected.
 - **status:** The status of the problem.
 - **severity:** The severity of the problem.
 - **problemUrl:** The console URL for the problem.
- **Problem update.** Emitted when the problem is updated with a new observation or when an existing observation is updated and the problem is subsequently updated; updates include a resolution or closure of the problem.
 - Detail Type: **"Application Insights Problem Updated"**
 - Detail:
 - **problemId:** The created problem ID.
 - **region:** The AWS Region where the problem was created.
 - **resourceGroupName:** The Resource Group for the registered application for which the problem was detected.
 - **status:** The status of the problem.
 - **severity:** The severity of the problem.
 - **problemUrl:** The console URL for the problem.

From the CloudWatch console, select **Rules** under **Events** in the left navigation pane. From the **Rules** page, select **Create rule**. Choose **Amazon CloudWatch Application Insights** from the **Service Name** dropdown list and choose the **Event Type**. Then, choose **Add target** and select the target and parameters, for example, an **SNS topic** or **Lambda function**.

Actions through AWS Systems Manager. CloudWatch Application Insights provides built-in integration with Systems Manager OpsCenter. If you choose to use this integration for your application, an OpsItem is created on the OpsCenter console for every problem detected with the application. From the OpsCenter console, you can view summarized information about the problem detected by CloudWatch Application Insights and pick a Systems Manager Automation runbook to take remedial actions or further identify Windows processes that are causing resource issues in your application.

Work with component configurations

A component configuration is a text file in JSON format that describes the configuration settings of the component. This section provides an example template fragment, descriptions of component configuration sections, and example component configurations.

Topics

- [Component configuration template fragment \(p. 634\)](#)
- [Component configuration sections \(p. 635\)](#)
- [Component configuration examples \(p. 640\)](#)

Component configuration template fragment

The following example shows a template fragment in JSON format.

```
{  
    "alarmMetrics" : [  
        list of alarm metrics  
    ],  
    "logs" : [  
        list of logs  
    ],  
    "windowsEvents" : [  
        list of windows events channels configurations  
    ],  
    "alarms" : [  
        list of CloudWatch alarms  
    ],  
    "jmxPrometheusExporter": {  
        JMX Prometheus Exporter configuration  
    },  
    "hanaPrometheusExporter": {  
        SAP HANA Prometheus Exporter configuration  
    },  
    "haClusterPrometheusExporter": {  
        HA Cluster Prometheus Exporter configuration  
    },  
    "subComponents" : [  
        {  
            "subComponentType" : "AWS::EC2::Instance" ...  
            component nested instances configuration  
        },  
        {  
            "subComponentType" : "AWS::EC2::Volume" ...  
            component nested volumes configuration  
        }  
    ]  
}
```

```
    ]  
}
```

Component configuration sections

A component configuration includes several major sections. Sections in a component configuration can be listed in any order.

- **alarmMetrics (optional)**

A list of [metrics \(p. 636\)](#) to monitor for the component. All component types can have an alarmMetrics section.

- **logs (optional)**

A list of [logs \(p. 636\)](#) to monitor for the component. Only EC2 instances can have a logs section.

- **subComponents (optional)**

Nested instance and volume subComponent configuration for the component. The following types of components can have nested instances and a subComponents section: ELB, ASG, custom-grouped EC2 instances , and EC2 instances.

- **alarms (optional)**

A list of [alarms \(p. 639\)](#) to monitor for the component. All component types can have an alarm section.

- **windowsEvents (optional)**

A list of [windows events \(p. 639\)](#) to monitor for the component. Only Windows on EC2 instances have a windowsEvents section.

- **JMXPrometheusExporter (optional)**

JMXPrometheus Exporter configuration.

- **hanaPrometheusExporter (optional)**

SAP HANA Prometheus Exporter configuration.

- **haClusterPrometheusExporter (optional)**

HA Cluster Prometheus Exporter configuration.

The following example shows the syntax for the **subComponents section fragment** in JSON format.

```
[  
  {  
    "subComponentType" : "AWS::EC2::Instance",  
    "alarmMetrics" : [  
      list of alarm metrics  
    ],  
    "logs" : [  
      list of logs  
    ],  
    "windowsEvents" : [  
      list of windows events channels configurations  
    ]  
  },  
  {  
    "subComponentType" : "AWS::EC2::Volume",  
    "alarmMetrics" : [  
      list of alarm metrics  
    ]  
  }]
```

```
        list of alarm metrics
    ]
}
```

Component configuration section properties

This section describes the properties of each component configuration section.

Sections

- [Metric \(p. 636\)](#)
- [Log \(p. 636\)](#)
- [JMX Prometheus Exporter \(p. 637\)](#)
- [HANA Prometheus Exporter \(p. 638\)](#)
- [HA Cluster Prometheus Exporter \(p. 638\)](#)
- [Windows Events \(p. 639\)](#)
- [Alarm \(p. 639\)](#)

Metric

Defines a metric to be monitored for the component.

JSON

```
{
  "alarmMetricName" : "monitoredMetricName",
  "monitor" : true/false
}
```

Properties

- **alarmMetricName (required)**

The name of the metric to be monitored for the component. For metrics supported by Application Insights, see [Logs and metrics supported by Amazon CloudWatch Application Insights \(p. 692\)](#).

- **monitor (optional)**

Boolean to indicate whether to monitor the metric. The default value is `true`.

Log

Defines a log to be monitored for the component.

JSON

```
{
  "logGroupName" : "logGroupName",
  "logPath" : "logPath",
  "logType" : "logType",
  "encoding" : "encodingType",
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see [CreateLogGroup](#).

- **logPath (required for EC2 instance components; not required for components that do not use CloudWatch Agent, such as AWS Lambda)**

The path of the logs to be monitored. The log path must be an absolute Windows system file path. For more information, see [CloudWatch Agent Configuration File: Logs Section](#).

- **logType (required)**

The log type decides the log patterns against which Application Insights analyzes the log. The log type is selected from the following:

- SQL_SERVER
- MYSQL
- MYSQL_SLOW_QUERY
- POSTGRESQL
- ORACLE_ALERT
- ORACLE_LISTENER
- IIS
- APPLICATION
- WINDOWS_EVENTS
- WINDOWS_EVENTS_ACTIVE_DIRECTORY
- WINDOWS_EVENTS_DNS
- WINDOWS_EVENTS_IIS
- WINDOWS_EVENTS_SHAREPOINT
- SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP
- SQL_SERVER_FAILOVER_CLUSTER_INSTANCE
- DEFAULT
- CUSTOM
- STEP_FUNCTION
- API_GATEWAY_ACCESS
- API_GATEWAY_EXECUTION
- SAP_HANA_LOGS
- SAP_HANA_TRACE
- SAP_HANA_HIGH_AVAILABILITY

- **encoding (optional)**

The type of encoding of the logs to be monitored. The specified encoding should be included in the list of [CloudWatch agent supported encodings](#). If not provided, CloudWatch Application Insights uses the default encoding of type utf-8, except for:

- SQL_SERVER: utf-16 encoding

- IIS: ascii encoding

- **monitor (optional)**

Boolean that indicates whether to monitor the logs. The default value is `true`.

JMX Prometheus Exporter

Defines the JMX Prometheus Exporter settings.

JSON

```
"JMXPrometheusExporter": {  
    "jmxURL" : "JMX URL",  
    "hostPort" : "The host and port",  
    "prometheusPort" : "Target port to emit Prometheus metrics"  
}
```

Properties

- **jmxURL (optional)**

A complete JMX URL to connect to.

- **hostPort (optional)**

The host and port to connect to through remote JMX. Only one of jmxURL and hostPort can be specified.

- **prometheusPort (optional)**

The target port to send Prometheus metrics to. If not specified, the default port 9404 is used.

HANA Prometheus Exporter

Defines the HANA Prometheus Exporter settings.

JSON

```
"hanaPrometheusExporter": {  
    "hanaSid": "SAP HANA SID",  
    "hanaPort": "HANA database port",  
    "hanaSecretName": "HANA secret name",  
    "prometheusPort": "Target port to emit Prometheus metrics"  
}
```

Properties

- **hanaSid**

The three-character SAP system ID (SID) of the SAP HANA system.

- **hanaPort**

The HANA database port by which the exporter will query HANA metrics.

- **hanaSecretName**

The AWS Secrets Manager secret that stores HANA monitoring user credentials. The HANA Prometheus exporter uses these credentials to connect to the database and query HANA metrics.

- **prometheusPort (optional)**

The target port to which Prometheus sends metrics. If not specified, the default port 9668 is used.

HA Cluster Prometheus Exporter

Defines the HA Cluster Prometheus Exporter settings.

JSON

```
"haClusterPrometheusExporter": {
```

```
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **prometheusPort (optional)**

The target port to which Prometheus sends metrics. If not specified, the default port 9664 is used.

Windows Events

Defines Windows Events to log.

JSON

```
{
  "logGroupName" : "logGroupName",
  "eventName" : "eventName",
  "eventLevels" : ["ERROR", "WARNING", "CRITICAL", "INFORMATION", "VERBOSE"],
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see [CreateLogGroup](#).

- **eventName (required)**

The type of Windows Events to log. It is equivalent to the Windows Event log channel name. For example, System, Security, CustomEventName, etc. This field is required for each type of Windows event to log.

- **eventLevels (required)**

The levels of event to log. You must specify each level to log. Possible values include INFORMATION, WARNING, ERROR, CRITICAL, and VERBOSE. This field is required for each type of Windows Event to log.

- **monitor (optional)**

Boolean that indicates whether to monitor the logs. The default value is true.

Alarm

Defines a CloudWatch alarm to be monitored for the component.

JSON

```
{
  "alarmName" : "monitoredAlarmName",
  "severity" : HIGH/MEDIUM/LOW
}
```

Properties

- **alarmName (required)**

The name of the CloudWatch alarm to be monitored for the component.

- **severity (optional)**

Indicates the degree of outage when the alarm goes off.

Component configuration examples

The following examples show component configurations in JSON format for relevant services.

Example component configurations

- [Amazon Elastic Compute Cloud \(EC2\) instance \(p. 640\)](#)
- [Amazon Relational Database Service instance \(p. 642\)](#)
- [Amazon Relational Database Service \(RDS\) Aurora MySQL \(p. 642\)](#)
- [Elastic Load Balancing \(ELB\) \(p. 642\)](#)
- [Application Elastic Load Balancing \(p. 643\)](#)
- [Amazon EC2 Auto Scaling \(ASG\) \(p. 644\)](#)
- [Amazon Simple Queue Service \(SQS\) \(p. 645\)](#)
- [Customer-grouped Amazon EC2 instances \(p. 645\)](#)
- [AWS Lambda Function \(p. 646\)](#)
- [Amazon DynamoDB table \(p. 647\)](#)
- [SQL Always On Availability Group \(p. 647\)](#)
- [SQL failover cluster instance \(p. 649\)](#)
- [RDS MariaDB and RDS MySQL \(p. 652\)](#)
- [RDS PostgreSQL \(p. 652\)](#)
- [Amazon S3 bucket \(p. 652\)](#)
- [AWS Step Functions \(p. 653\)](#)
- [API Gateway REST API stages \(p. 653\)](#)
- [Java \(p. 654\)](#)
- [RDS Oracle \(p. 654\)](#)
- [Amazon Elastic Container Service \(Amazon ECS\) \(p. 654\)](#)
- [Amazon ECS service \(p. 657\)](#)
- [Amazon ECS task \(p. 660\)](#)
- [Amazon EKS cluster \(p. 660\)](#)
- [Kubernetes on Amazon EC2 \(p. 662\)](#)
- [Amazon FSx \(p. 665\)](#)
- [Amazon SNS topic \(p. 665\)](#)
- [SAP HANA on Amazon EC2 \(p. 666\)](#)
- [SAP HANA High Availability on Amazon EC2 \(p. 667\)](#)

Amazon Elastic Compute Cloud (EC2) instance

The following example shows a component configuration in JSON format for an Amazon EC2 instance.

Important

When an Amazon EC2 instance enters a stopped state, it is removed from monitoring.

When it returns to a running state, it is added to the list of **Unmonitored components** on the **Application details** page of the CloudWatch Application Insights console. If automatic

monitoring of new resources is enabled for the application, the instance is added to the list of **Monitored components**. However, the logs and metrics are set to the default for the workload. The previous log and metrics configuration is not saved.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    },
    {
      "alarmMetricName" : "StatusCheckFailed"
    }
  ],
  "logs" : [
    {
      "logGroupName" : "my_log_group",
      "logPath" : "C:\\LogFolder\\*",
      "logType" : "APPLICATION",
      "monitor" : true
    },
    {
      "logGroupName" : "my_log_group_2",
      "logPath" : "C:\\LogFolder2\\*",
      "logType" : "IIS",
      "encoding" : "utf-8"
    }
  ],
  "windowsEvents" : [
    {
      "logGroupName" : "my_log_group_3",
      "eventName" : "Application",
      "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
      "monitor" : true
    },
    {
      "logGroupName" : "my_log_group_4",
      "eventName" : "System",
      "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
      "monitor" : true
    }
  ],
  "alarms" : [
    {
      "alarmName" : "my_instance_alarm_1",
      "severity" : "HIGH"
    },
    {
      "alarmName" : "my_instance_alarm_2",
      "severity" : "LOW"
    }
  ],
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Volume",
      "alarmMetrics" : [
        {
          "alarmMetricName" : "VolumeQueueLength",
          "monitor" : "true"
        },
        {
          "alarmMetricName" : "VolumeThroughputPercentage",
          "monitor" : "true"
        },
        {
          "alarmMetricName" : "BurstBalance",
          "monitor" : "true"
        }
      ]
    }
  ]
}
```

```
    }]
}
```

Amazon Relational Database Service instance

The following example shows a component configuration in JSON format for an Amazon RDS instance.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    },
    {
      "alarmMetricName" : "WriteThroughput",
      "monitor" : false
    }
  ],
  "alarms" : [
    {
      "alarmName" : "my_rds_instance_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon Relational Database Service (RDS) Aurora MySQL

The following example shows a component configuration in JSON format for Amazon RDS Aurora MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CommitLatency",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

Elastic Load Balancing (ELB)

The following example shows a component configuration in JSON format for Elastic Load Balancing.

```
{
  "alarmMetrics" : [
    {
```

```

        "alarmMetricName" : "EstimatedALBActiveConnectionCount",
    }, {
        "alarmMetricName" : "HTTPCode_Backend_5XX"
    },
],
"subComponents" : [
{
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
        {
            "alarmMetricName" : "CPUUtilization",
        }, {
            "alarmMetricName" : "StatusCheckFailed"
        }
    ],
    "logs" : [
        {
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION",
        }
    ],
    "windowsEvents" : [
        {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
            "monitor" : true
        }
    ]
},
{
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
        {
            "alarmMetricName" : "VolumeQueueLength",
        }, {
            "alarmMetricName" : "BurstBalance"
        }
    ]
},
"alarms" : [
{
    "alarmName" : "my_elb_alarm",
    "severity" : "HIGH"
}
]
}
]
}

```

Application Elastic Load Balancing

The following example shows a component configuration in JSON format for Application Elastic Load Balancing.

```
{
"alarmMetrics" : [
{
    "alarmMetricName" : "ActiveConnectionCount",
}, {
    "alarmMetricName" : "TargetResponseTime"
}
],
"subComponents" : [

```

```
{
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
        {
            "alarmMetricName" : "CPUUtilization",
        },
        {
            "alarmMetricName" : "StatusCheckFailed"
        }
    ],
    "logs" : [
        {
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION",
        }
    ],
    "windowsEvents" : [
        {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
        }
    ]
},
{
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
        {
            "alarmMetricName" : "VolumeQueueLength",
        },
        {
            "alarmMetricName" : "BurstBalance"
        }
    ]
},
{
    "alarms" : [
        {
            "alarmName" : "my_alb_alarm",
            "severity" : "LOW"
        }
    ]
}
```

Amazon EC2 Auto Scaling (ASG)

The following example shows a component configuration in JSON format for Amazon EC2 Auto Scaling (ASG).

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "CPUCreditBalance",
        },
        {
            "alarmMetricName" : "EBSIOBalance%"
        }
    ],
    "subComponents" : [
        {
            "subComponentType" : "AWS::EC2::Instance",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "CPUUtilization",
                },
                {
                    "alarmMetricName" : "StatusCheckFailed"
                }
            ]
        }
    ]
}
```

```

        }
    ],
    "logs" : [
        {
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION",
        }
    ],
    "windowsEvents" : [
        {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
        }
    ]
},
{
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
        {
            "alarmMetricName" : "VolumeQueueLength",
        },
        {
            "alarmMetricName" : "BurstBalance"
        }
    ]
},
"alarms" : [
    {
        "alarmName" : "my_asg_alarm",
        "severity" : "LOW"
    }
]
}

```

Amazon Simple Queue Service (SQS)

The following example shows a component configuration in JSON format for Amazon Simple Queue Service.

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "ApproximateAgeOfOldestMessage"
        },
        {
            "alarmMetricName" : "NumberOfEmptyReceives"
        }
    ],
    "alarms" : [
        {
            "alarmName" : "my_sqs_alarm",
            "severity" : "MEDIUM"
        }
    ]
}
```

Customer-grouped Amazon EC2 instances

The following example shows a component configuration in JSON format for customer-grouped Amazon EC2 instances.

```
{
}
```

```

"subComponents" : [
  {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUUtilization",
        }, {
        "alarmMetricName" : "StatusCheckFailed"
      }
    ],
    "logs" : [
      {
        "logGroupName" : "my_log_group",
        "logPath" : "C:\\LogFolder\\*",
        "logType" : "APPLICATION",
      }
    ],
    "windowsEvents" : [
      {
        "logGroupName" : "my_log_group_2",
        "eventName" : "Application",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
      }
    ]
  },
  {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
      {
        "alarmMetricName" : "VolumeQueueLength",
        }, {
        "alarmMetricName" : "BurstBalance"
      }
    ]
  },
  "alarms" : [
    {
      "alarmName" : "my_alarm",
      "severity" : "MEDIUM"
    }
  ]
]
}

```

AWS Lambda Function

The following example shows a component configuration in JSON format for AWS Lambda Function.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "Errors",
      "monitor": true
    },
    {
      "alarmMetricName": "Throttles",
      "monitor": true
    },
    {
      "alarmMetricName": "IteratorAge",
      "monitor": true
    },
    {
      "alarmMetricName": "Duration",
      "monitor": true
    }
  ]
}
```

```

        },
        ],
        "logs": [
        {
            "logType": "DEFAULT",
            "monitor": true
        }
    ]
}

```

Amazon DynamoDB table

The following example shows a component configuration in JSON format for Amazon DynamoDB table.

```

{
    "alarmMetrics": [
    {
        "alarmMetricName": "SystemErrors",
        "monitor": false
    },
    {
        "alarmMetricName": "UserErrors",
        "monitor": false
    },
    {
        "alarmMetricName": "ConsumedReadCapacityUnits",
        "monitor": false
    },
    {
        "alarmMetricName": "ConsumedWriteCapacityUnits",
        "monitor": false
    },
    {
        "alarmMetricName": "ReadThrottleEvents",
        "monitor": false
    },
    {
        "alarmMetricName": "WriteThrottleEvents",
        "monitor": false
    },
    {
        "alarmMetricName": "ConditionalCheckFailedRequests",
        "monitor": false
    },
    {
        "alarmMetricName": "TransactionConflict",
        "monitor": false
    }
],
    "logs": []
}

```

SQL Always On Availability Group

The following example shows a component configuration in JSON format for SQL Always On Availability Group.

```

{
    "subComponents" : [ {
        "subComponentType" : "AWS::EC2::Instance",
        "alarmMetrics" : [ {

```

```
        "alarmMetricName" : "CPUUtilization",
        "monitor" : true
    }, {
        "alarmMetricName" : "StatusCheckFailed",
        "monitor" : true
    }, {
        "alarmMetricName" : "Processor % Processor Time",
        "monitor" : true
    }, {
        "alarmMetricName" : "Memory % Committed Bytes In Use",
        "monitor" : true
    }, {
        "alarmMetricName" : "Memory Available Mbytes",
        "monitor" : true
    }, {
        "alarmMetricName" : "Paging File % Usage",
        "monitor" : true
    }, {
        "alarmMetricName" : "System Processor Queue Length",
        "monitor" : true
    }, {
        "alarmMetricName" : "Network Interface Bytes Total/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "PhysicalDisk % Disk Time",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:General Statistics User Connections",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
        "monitor" : true
    }, {
        "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
        "monitor" : true
    }, {
```

```
"alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
"monitor" : true
}, {
"alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
"monitor" : true
}, {
"alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
"monitor" : true
} ],
"windowsEvents" : [ {
"logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
"eventName" : "Application",
"eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
"monitor" : true
}, {
"logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
"eventName" : "System",
"eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
"monitor" : true
}, {
"logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
"eventName" : "Security",
"eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
"monitor" : true
} ],
"logs" : [ {
"logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
"logPath" : "C:\Program Files\Microsoft SQL Server\MSSQL**.MSSQLSERVER\MSSQL\Log
\\ERRORLOG",
"logType" : "SQL_SERVER",
"monitor" : true,
"encoding" : "utf-8"
} ]
}, {
"subComponentType" : "AWS::EC2::Volume",
"alarmMetrics" : [ {
"alarmMetricName" : "VolumeReadBytes",
"monitor" : true
}, {
"alarmMetricName" : "VolumeWriteBytes",
"monitor" : true
}, {
"alarmMetricName" : "VolumeReadOps",
"monitor" : true
}, {
"alarmMetricName" : "VolumeWriteOps",
"monitor" : true
}, {
"alarmMetricName" : "VolumeQueueLength",
"monitor" : true
}, {
"alarmMetricName" : "VolumeThroughputPercentage",
"monitor" : true
}, {
"alarmMetricName" : "BurstBalance",
"monitor" : true
} ]
} ]
```

SQL failover cluster instance

The following example shows a component configuration in JSON format for SQL failover cluster instance.

```
{  
    "subComponents" : [ {  
        "subComponentType" : "AWS::EC2::Instance",  
        "alarmMetrics" : [ {  
            "alarmMetricName" : "CPUUtilization",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "StatusCheckFailed",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Processor % Processor Time",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Memory % Committed Bytes In Use",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Memory Available Mbytes",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Paging File % Usage",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "System Processor Queue Length",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Network Interface Bytes Total/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "PhysicalDisk % Disk Time",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Bytes Received/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Normal Messages Queue Length/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Urgent Message Queue Length/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Reconnect Count",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Unacknowledged Message Queue Length/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Messages Outstanding",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Messages Sent/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Database Update Messages/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Update Messages/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Flushes/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Crypto Checkpoints Saved/sec",  
            "monitor" : true  
        }, {  
            "alarmMetricName" : "Crypto Checkpoints Restored/sec",  
            "monitor" : true  
        } ]  
    } ]  
}
```

```

        "monitor" : true
    }, {
        "alarmMetricName" : "Registry Checkpoints Restored/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "Registry Checkpoints Saved/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "Cluster API Calls/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "Resource API Calls/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "Cluster Handles/sec",
        "monitor" : true
    }, {
        "alarmMetricName" : "Resource Handles/sec",
        "monitor" : true
    } ],
"windowsEvents" : [ {
    "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
    "eventName" : "Application",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
}, {
    "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
    "eventName" : "System",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
    "monitor" : true
}, {
    "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
    "eventName" : "Security",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
} ],
"logs" : [ {
    "logGroupName" : "SQL_SERVER_FAILOVER_CLUSTER_INSTANCE-<RESOURCE_GROUP_NAME>",
    "logPath" : "\\\amznfsxjmzbykwn.mydomain.aws\\SQLDB\\MSSQL**.MSSQLSERVER\\MSSQL\\Log\\ERRORLOG",
    "logType" : "SQL_SERVER",
    "monitor" : true,
    "encoding" : "utf-8"
} ]
}, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
        "alarmMetricName" : "VolumeReadBytes",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeWriteBytes",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeReadOps",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeWriteOps",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeQueueLength",
        "monitor" : true
    }, {
        "alarmMetricName" : "VolumeThroughputPercentage",
        "monitor" : true
    }, {
        "alarmMetricName" : "BurstBalance",
        "monitor" : true
    }
]
}

```

```
        "monitor" : true
    } ]
}
}
```

RDS MariaDB and RDS MySQL

The following example shows a component configuration in JSON format for RDS MariaDB and RDS MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

RDS PostgreSQL

The following example shows a component configurations in JSON format for RDS PostgreSQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "POSTGRESQL",
      "monitor": true
    }
  ]
}
```

Amazon S3 bucket

The following example shows a component configurations in JSON format for Amazon S3 bucket.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ReplicationLatency",
      "monitor" : true
    },
    {
      "alarmMetricName" : "5xxErrors",
      "monitor" : true
    }
  ]
}
```

```
        "monitor" : true
    }, {
        "alarmMetricName" : "BytesDownloaded"
        "monitor" : true
    }
]
}
```

AWS Step Functions

The following example shows a component configurations in JSON format for AWS Step Functions.

```
{
    "alarmMetrics": [
        {
            "alarmMetricName": "ExecutionsFailed",
            "monitor": true
        },
        {
            "alarmMetricName": "LambdaFunctionsFailed",
            "monitor": true
        },
        {
            "alarmMetricName": "ProvisionedRefillRate",
            "monitor": true
        }
    ],
    "logs": [
        {
            "logGroupName": "/aws/states/HelloWorld-Logs",
            "logType": "STEP_FUNCTION",
            "monitor": true,
        }
    ]
}
```

API Gateway REST API stages

The following example shows a component configuration in JSON format for API Gateway REST API stages.

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "4XXError",
            "monitor" : true
        },
        {
            "alarmMetricName" : "5XXError",
            "monitor" : true
        }
    ],
    "logs" : [
        {
            "logType" : "API_GATEWAY_EXECUTION",
            "monitor" : true
        },
        {
            "logType" : "API_GATEWAY_ACCESS",
            "monitor" : true
        },
        {
            "logType" : "API_GATEWAY_INTEGRATION_FAILURE_RESOLUTION",
            "monitor" : true
        }
    ]
}
```

```
}
```

Java

The following example shows a component configuration in JSON format for Java.

```
{
    "alarmMetrics" : [ {
        "alarmMetricName" : "java_lang_threading_threadcount",
        "monitor" : true
    },
    {
        "alarmMetricName" : "java_lang_memory_heapmemoryusage_used",
        "monitor" : true
    },
    {
        "alarmMetricName" : "java_lang_memory_heapmemoryusage_committed",
        "monitor" : true
    }],
    "logs" : [ ],
    "JMXPrometheusExporter": {
        "hostPort": "8686",
        "prometheusPort": "9404"
    }
}
```

Note

Application Insights does not support configuring authentication for Prometheus JMX exporter. For information about how to set up authentication, see the [Prometheus JMX exporter example configuration](#).

RDS Oracle

The following example shows a component configuration in JSON format for RDS Oracle.

```
{
    "alarmMetrics": [
        {
            "alarmMetricName": "CPUUtilization",
            "monitor": true
        }
    ],
    "logs": [
        {
            "logType": "ORACLE_ALERT",
            "monitor": true,
        },
        {
            "logType": "ORACLE_LISTENER",
            "monitor": false
        }
    ]
}
```

Amazon Elastic Container Service (Amazon ECS)

The following example shows a component configuration in JSON format for Amazon Elastic Container Service (Amazon ECS).

```
{
```

```
"alarmMetrics": [
    {
        "alarmMetricName": "CpuUtilized",
        "monitor": true
    },
    {
        "alarmMetricName": "MemoryUtilized",
        "monitor": true
    },
    {
        "alarmMetricName": "NetworkRxBytes",
        "monitor": true
    },
    {
        "alarmMetricName": "NetworkTxBytes",
        "monitor": true
    },
    {
        "alarmMetricName": "RunningTaskCount",
        "monitor": true
    },
    {
        "alarmMetricName": "PendingTaskCount",
        "monitor": true
    },
    {
        "alarmMetricName": "StorageReadBytes",
        "monitor": true
    },
    {
        "alarmMetricName": "StorageWriteBytes",
        "monitor": true
    }
],
"logs": [
    {
        "logGroupName": "/ecs/my-task-definition",
        "logType": "APPLICATION",
        "monitor": true
    }
],
"subComponents": [
    {
        "subComponentType": "AWS::ElasticLoadBalancing::LoadBalancer",
        "alarmMetrics": [
            {
                "alarmMetricName": "HTTPCode_Backend_4XX",
                "monitor": true
            },
            {
                "alarmMetricName": "HTTPCode_Backend_5XX",
                "monitor": true
            },
            {
                "alarmMetricName": "Latency",
                "monitor": true
            },
            {
                "alarmMetricName": "SurgeQueueLength",
                "monitor": true
            },
            {
                "alarmMetricName": "UnHealthyHostCount",
                "monitor": true
            }
        ]
    }
]
```

```

},
{
    "subComponentType": "AWS::ElasticLoadBalancingV2::LoadBalancer",
    "alarmMetrics": [
        {
            "alarmMetricName": "HTTPCode_Target_4XX_Count",
            "monitor": true
        },
        {
            "alarmMetricName": "HTTPCode_Target_5XX_Count",
            "monitor": true
        },
        {
            "alarmMetricName": "TargetResponseTime",
            "monitor": true
        },
        {
            "alarmMetricName": "UnHealthyHostCount",
            "monitor": true
        }
    ]
},
{
    "subComponentType": "AWS::EC2::Instance",
    "alarmMetrics": [
        {
            "alarmMetricName": "CPUUtilization",
            "monitor": true
        },
        {
            "alarmMetricName": "StatusCheckFailed",
            "monitor": true
        },
        {
            "alarmMetricName": "disk_used_percent",
            "monitor": true
        },
        {
            "alarmMetricName": "mem_used_percent",
            "monitor": true
        }
    ],
    "logs": [
        {
            "logGroupName": "my_log_group",
            "logPath": "/mylog/path",
            "logType": "APPLICATION",
            "monitor": true
        }
    ],
    "windowsEvents": [
        {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [
                "ERROR",
                "WARNING",
                "CRITICAL"
            ],
            "monitor": true
        }
    ]
},
{
    "subComponentType": "AWS::EC2::Volume",
    "alarmMetrics": [

```

```
{
    "alarmMetricName": "VolumeQueueLength",
    "monitor": "true"
},
{
    "alarmMetricName": "VolumeThroughputPercentage",
    "monitor": "true"
},
{
    "alarmMetricName": "BurstBalance",
    "monitor": "true"
}
]
}
}
```

Note

- The `subComponents` section of `AWS::EC2::Instance` and `AWS::EC2::Volume` applies only to Amazon ECS clusters with ECS service or ECS task running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon ECS service

The following example shows a component configuration in JSON format for Amazon ECS service.

```
{
    "alarmMetrics": [
        {
            "alarmMetricName": "CPUUtilization",
            "monitor": true
        },
        {
            "alarmMetricName": "MemoryUtilization",
            "monitor": true
        },
        {
            "alarmMetricName": "CpuUtilized",
            "monitor": true
        },
        {
            "alarmMetricName": "MemoryUtilized",
            "monitor": true
        },
        {
            "alarmMetricName": "NetworkRxBytes",
            "monitor": true
        },
        {
            "alarmMetricName": "NetworkTxBytes",
            "monitor": true
        },
        {
            "alarmMetricName": "RunningTaskCount",
            "monitor": true
        },
        {
            "alarmMetricName": "PendingTaskCount",
            "monitor": true
        }
    ]
}
```

```
{
    "alarmMetricName": "StorageReadBytes",
    "monitor": true
},
{
    "alarmMetricName": "StorageWriteBytes",
    "monitor": true
}
],
"logs": [
    {
        "logGroupName": "/ecs/my-task-definition",
        "logType": "APPLICATION",
        "monitor": true
    }
],
"subComponents": [
    {
        "subComponentType": "AWS::ElasticLoadBalancing::LoadBalancer",
        "alarmMetrics": [
            {
                "alarmMetricName": "HTTPCode_Backend_4XX",
                "monitor": true
            },
            {
                "alarmMetricName": "HTTPCode_Backend_5XX",
                "monitor": true
            },
            {
                "alarmMetricName": "Latency",
                "monitor": true
            },
            {
                "alarmMetricName": "SurgeQueueLength",
                "monitor": true
            },
            {
                "alarmMetricName": "UnHealthyHostCount",
                "monitor": true
            }
        ]
    },
    {
        "subComponentType": "AWS::ElasticLoadBalancingV2::LoadBalancer",
        "alarmMetrics": [
            {
                "alarmMetricName": "HTTPCode_Target_4XX_Count",
                "monitor": true
            },
            {
                "alarmMetricName": "HTTPCode_Target_5XX_Count",
                "monitor": true
            },
            {
                "alarmMetricName": "TargetResponseTime",
                "monitor": true
            },
            {
                "alarmMetricName": "UnHealthyHostCount",
                "monitor": true
            }
        ]
    },
    {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [

```

```
{
    "alarmMetrics": [
        {
            "alarmMetricName": "CPUUtilization",
            "monitor": true
        },
        {
            "alarmMetricName": "StatusCheckFailed",
            "monitor": true
        },
        {
            "alarmMetricName": "disk_used_percent",
            "monitor": true
        },
        {
            "alarmMetricName": "mem_used_percent",
            "monitor": true
        }
    ],
    "logs": [
        {
            "logGroupName": "my_log_group",
            "logPath": "/mylog/path",
            "logType": "APPLICATION",
            "monitor": true
        }
    ],
    "windowsEvents": [
        {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [
                "ERROR",
                "WARNING",
                "CRITICAL"
            ],
            "monitor": true
        }
    ]
},
{
    "subComponentType": "AWS::EC2::Volume",
    "alarmMetrics": [
        {
            "alarmMetricName": "VolumeQueueLength",
            "monitor": "true"
        },
        {
            "alarmMetricName": "VolumeThroughputPercentage",
            "monitor": "true"
        },
        {
            "alarmMetricName": "BurstBalance",
            "monitor": "true"
        }
    ]
}
}
```

Note

- The `subComponents` section of `AWS::EC2::Instance` and `AWS::EC2::Volume` applies only to Amazon ECS running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon ECS task

The following example shows a component configuration in JSON format for Amazon ECS task.

```
{  
    "logs": [  
        {  
            "logGroupName": "/ecs/my-task-definition",  
            "logType": "APPLICATION",  
            "monitor": true  
        }  
    ]  
}
```

Amazon EKS cluster

The following example shows a component configuration in JSON format for Amazon EKS cluster.

```
{  
    "alarmMetrics": [  
        {  
            "alarmMetricName": "cluster_failed_node_count",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "node_cpu_reserved_capacity",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "node_cpu_utilization",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "node_filesystem_utilization",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "node_memory_reserved_capacity",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "node_memory_utilization",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "node_network_total_bytes",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "pod_cpu_reserved_capacity",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "pod_cpu_utilization",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "pod_cpu_utilization_over_pod_limit",  
            "monitor": true  
        },  
        {  
            "alarmMetricName": "pod_memory_reserved_capacity",  
            "monitor": true  
        }  
    ]  
}
```

```

},
{
    "alarmMetricName": "pod_memory_utilization",
    "monitor": true
},
{
    "alarmMetricName": "pod_memory_utilization_over_pod_limit",
    "monitor": true
},
{
    "alarmMetricName": "pod_network_rx_bytes",
    "monitor": true
},
{
    "alarmMetricName": "pod_network_tx_bytes",
    "monitor": true
}
],
"logs": [
    {
        "logGroupName": "/aws/containerinsights/kubernetes/application",
        "logType": "APPLICATION",
        "monitor": true,
        "encoding": "utf-8"
    }
],
"subComponents": [
    {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
            {
                "alarmMetricName": "CPUUtilization",
                "monitor": true
            },
            {
                "alarmMetricName": "StatusCheckFailed",
                "monitor": true
            },
            {
                "alarmMetricName": "disk_used_percent",
                "monitor": true
            },
            {
                "alarmMetricName": "mem_used_percent",
                "monitor": true
            }
        ],
        "logs": [
            {
                "logGroupName": "APPLICATION-KubernetesClusterOnEC2-IAD",
                "logPath": "",
                "logType": "APPLICATION",
                "monitor": true,
                "encoding": "utf-8"
            }
        ],
        "windowsEvents": [
            {
                "logGroupName": "my_log_group_2",
                "eventName": "Application",
                "eventLevels": [
                    "ERROR",
                    "WARNING",
                    "CRITICAL"
                ],
                "monitor": true
            }
        ]
    }
]
}

```

```

        }
    ],
{
    "subComponentType": "AWS::AutoScaling::AutoScalingGroup",
    "alarmMetrics": [
        {
            "alarmMetricName": "CPUCreditBalance",
            "monitor": true
        },
        {
            "alarmMetricName": "EBSIOBalance%",
            "monitor": true
        }
    ]
},
{
    "subComponentType": "AWS::EC2::Volume",
    "alarmMetrics": [
        {
            "alarmMetricName": "VolumeReadBytes",
            "monitor": true
        },
        {
            "alarmMetricName": "VolumeWriteBytes",
            "monitor": true
        },
        {
            "alarmMetricName": "VolumeReadOps",
            "monitor": true
        },
        {
            "alarmMetricName": "VolumeWriteOps",
            "monitor": true
        },
        {
            "alarmMetricName": "VolumeQueueLength",
            "monitor": true
        },
        {
            "alarmMetricName": "BurstBalance",
            "monitor": true
        }
    ]
}
}

```

Note

- The subComponents section of AWS::EC2::Instance, AWS::EC2::Volume, and AWS::AutoScaling::AutoScalingGroup applies only to Amazon EKS cluster running on the EC2 launch type.
- The windowsEvents section of AWS::EC2::Instance in subComponents applies only to Windows running on Amazon EC2 instances.

Kubernetes on Amazon EC2

The following example shows a component configuration in JSON format for Kubernetes on Amazon EC2.

```
{
}
```

```
"alarmMetrics": [
    {
        "alarmMetricName": "cluster_failed_node_count",
        "monitor": true
    },
    {
        "alarmMetricName": "node_cpu_reserved_capacity",
        "monitor": true
    },
    {
        "alarmMetricName": "node_cpu_utilization",
        "monitor": true
    },
    {
        "alarmMetricName": "node_filesystem_utilization",
        "monitor": true
    },
    {
        "alarmMetricName": "node_memory_reserved_capacity",
        "monitor": true
    },
    {
        "alarmMetricName": "node_memory_utilization",
        "monitor": true
    },
    {
        "alarmMetricName": "node_network_total_bytes",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_cpu_reserved_capacity",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_cpu_utilization",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_memory_reserved_capacity",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_memory_utilization",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_memory_utilization_over_pod_limit",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_network_rx_bytes",
        "monitor": true
    },
    {
        "alarmMetricName": "pod_network_tx_bytes",
        "monitor": true
    }
],
"logs": [
    {
        "logGroupName": "/aws/containerinsights/kubernetes/application",
        "logType": "APPLICATION",
    }
]
```

```
        "monitor":true,
        "encoding":"utf-8"
    }
],
"subComponents":[
{
    "subComponentType":"AWS::EC2::Instance",
    "alarmMetrics":[
        {
            "alarmMetricName":"CPUUtilization",
            "monitor":true
        },
        {
            "alarmMetricName":"StatusCheckFailed",
            "monitor":true
        },
        {
            "alarmMetricName":"disk_used_percent",
            "monitor":true
        },
        {
            "alarmMetricName":"mem_used_percent",
            "monitor":true
        }
    ],
    "logs":[
        {
            "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
            "logPath":"",
            "logType":"APPLICATION",
            "monitor":true,
            "encoding":"utf-8"
        }
    ]
},
{
    "subComponentType":"AWS::EC2::Volume",
    "alarmMetrics":[
        {
            "alarmMetricName":"VolumeReadBytes",
            "monitor":true
        },
        {
            "alarmMetricName":"VolumeWriteBytes",
            "monitor":true
        },
        {
            "alarmMetricName":"VolumeReadOps",
            "monitor":true
        },
        {
            "alarmMetricName":"VolumeWriteOps",
            "monitor":true
        },
        {
            "alarmMetricName":"VolumeQueueLength",
            "monitor":true
        },
        {
            "alarmMetricName":"BurstBalance",
            "monitor":true
        }
    ]
}
]
```

```
}
```

Amazon FSx

The following example shows a component configuration in JSON format for Amazon FSx.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DataReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "FreeStorageCapacity",
      "monitor": true
    }
  ]
}
```

Amazon SNS topic

The following example shows a component configuration in JSON format for Amazon SNS topic.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NumberOfNotificationsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-InvalidAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-NoMessageAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFailedToRedriveToDlq",
      "monitor": true
    }
  ]
}
```

SAP HANA on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA on Amazon EC2.

```
{  
    "subComponents": [  
        {  
            "subComponentType": "AWS::EC2::Instance",  
            "alarmMetrics": [  
                {  
                    "alarmMetricName": "hanadb_server_startup_time_variations_seconds",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_level_5_alerts_count",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_level_4_alerts_count",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_out_of_memory_events_count",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_max_trigger_read_ratio_percent",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_table_allocation_limit_used_percent",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_cpu_usage_percent",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_plan_cache_hit_ratio_percent",  
                    "monitor": true  
                },  
                {  
                    "alarmMetricName": "hanadb_last_data_backup_age_days",  
                    "monitor": true  
                }  
            ],  
            "logs": [  
                {  
                    "logGroupName": "SAP_HANA_TRACE-my-resourge-group",  
                    "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",  
                    "logType": "SAP_HANA_TRACE",  
                    "monitor": true,  
                    "encoding": "utf-8"  
                },  
                {  
                    "logGroupName": "SAP_HANA_LOGS-my-resource-group",  
                    "logPath": "/usr/sap/HDB/HDB00/*/trace/*.log",  
                    "logType": "SAP_HANA_LOGS",  
                    "monitor": true,  
                    "encoding": "utf-8"  
                }  
            ]  
        }  
    ],  
}
```

```

    "hanaPrometheusExporter": {
        "hanaSid": "HDB",
        "hanaPort": "30013",
        "hanaSecretName": "HANA_DB_CREDS",
        "prometheusPort": "9668"
    }
}

```

SAP HANA High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA High Availability on Amazon EC2.

```

{
    "subComponents": [
        {
            "subComponentType": "AWS::EC2::Instance",
            "alarmMetrics": [
                {
                    "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
                    "monitor": true
                },
                {
                    "alarmMetricName": "hanadb_level_5_alerts_count",
                    "monitor": true
                },
                {
                    "alarmMetricName": "hanadb_level_4_alerts_count",
                    "monitor": true
                },
                {
                    "alarmMetricName": "hanadb_out_of_memory_events_count",
                    "monitor": true
                },
                {
                    "alarmMetricName": "ha_cluster_pacemaker_stonith_enabled",
                    "monitor": true
                }
            ],
            "logs": [
                {
                    "logGroupName": "SAP_HANA_TRACE-my-resource-group",
                    "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
                    "logType": "SAP_HANA_TRACE",
                    "monitor": true,
                    "encoding": "utf-8"
                },
                {
                    "logGroupName": "SAP_HANA_HIGH_AVAILABILITY-my-resource-group",
                    "logPath": "/var/log/pacemaker/pacemaker.log",
                    "logType": "SAP_HANA_HIGH_AVAILABILITY",
                    "monitor": true,
                    "encoding": "utf-8"
                }
            ]
        },
        "hanaPrometheusExporter": {
            "hanaSid": "HDB",
            "hanaPort": "30013",
            "hanaSecretName": "HANA_DB_CREDS",
            "prometheusPort": "9668"
        },
        "haClusterPrometheusExporter": {

```

```
        "prometheusPort": "9664"
    }
```

Create and configure CloudWatch Application Insights monitoring using CloudFormation templates

You can add Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from AWS CloudFormation templates.

This section provides sample AWS CloudFormation templates in both JSON and YAML formats to help you create and configure Application Insights monitoring.

To view the Application Insights resource and property reference in the *AWS CloudFormation User Guide*, see [ApplicationInsights resource type reference](#).

Sample templates

- [Create an Application Insights application for the entire AWS CloudFormation stack \(p. 668\)](#)
- [Create an Application Insights application with detailed settings \(p. 670\)](#)
- [Create an Application Insights application with CUSTOM mode component configuration \(p. 672\)](#)
- [Create an Application Insights application with DEFAULT mode component configuration \(p. 674\)](#)
- [Create an Application Insights application with DEFAULT_WITH_OVERWRITE mode component configuration \(p. 675\)](#)

Create an Application Insights application for the entire AWS CloudFormation stack

To apply the following template, you must create AWS resources and one or more resource groups from which to create Application Insights applications to monitor those resources. For more information, see [Getting started with AWS Resource Groups](#).

The first two parts of the following template specify a resource and a resource group. The last part of the template creates an Application Insights application for the resource group, but does not configure the application or apply monitoring. For more information, see the [CreateApplication](#) command details in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Resource Group stack",
    "Resources": {
        "EC2Instance": {
            "Type": "AWS::EC2::Instance",
            "Properties": {
                "ImageId" : "ami-abcd1234efgh5678i",
                "SecurityGroupIds" : ["sg-abcd1234"]
            }
        },
        ...
    },
    "ResourceGroup": {
```

```

        "Type": "AWS::ResourceGroups::Group",
        "Properties": {
            "Name": "my_resource_group"
        }
    },
    "AppInsightsApp": {
        "Type": "AWS::ApplicationInsights::Application",
        "Properties": {
            "ResourceGroupName": "my_resource_group"
        },
        "DependsOn" : "ResourceGroup"
    }
}
}

```

Template in YAML format

```

---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Resource Group stack
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-abcd1234efgh5678i
      SecurityGroupIds:
        - sg-abcd1234
  ...
  ResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name: my_resource_group
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      DependsOn: ResourceGroup

```

The following template section applies the default monitoring configuration to the Application Insights application. For more information, see the [CreateApplication](#) command details in the *Amazon CloudWatch Application Insights API Reference*.

When `AutoConfigurationEnabled` is set to `true`, all components of the application are configured with the recommended monitoring settings for the `DEFAULT` application tier. For more information about these settings and tiers, see [DescribeComponentConfigurationRecommendation](#) and [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Test Application Insights Application stack",
  "Resources": {
    "AppInsightsApp": {
      "Type": "AWS::ApplicationInsights::Application",
      "Properties": {
        "ResourceGroupName": "my_resource_group",
        "AutoConfigurationEnabled": true
      }
    }
  }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Application Insights Application stack
Resources:
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      AutoConfigurationEnabled: true
```

Create an Application Insights application with detailed settings

The following template performs these actions:

- Creates an Application Insights application with CloudWatch Events notification and OpsCenter enabled. For more information, see the [CreateApplication](#) command details in the *Amazon CloudWatch Application Insights API Reference*.
- Tags the application with two tags, one of which has no tag values. For more information, see [TagResource](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two custom instance group components. For more information, see [CreateComponent](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two log pattern sets. For more information, see [CreateLogPattern](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Sets AutoConfigurationEnabled to true, which configures all components of the application with the recommended monitoring settings for the DEFAULT tier. For more information, see [DescribeComponentConfigurationRecommendation](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
  "Type": "AWS::ApplicationInsights::Application",
  "Properties": {
    "ResourceGroupName": "my_resource_group",
    "CWEEnabled": true,
    "OpsCenterEnabled": true,
    "OpsItemSNSTopicArn": "arn:aws:sns:us-east-1:123456789012:my_topic",
    "AutoConfigurationEnabled": true,
    "Tags": [
      {
        "Key": "key1",
        "Value": "value1"
      },
      {
        "Key": "key2",
        "Value": ""
      }
    ],
    "CustomComponents": [
      {
        "ComponentName": "test_component_1",
        "ResourceList": [
          "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
        ]
      }
    ]
  }
}
```

```
        },
        {
            "ComponentName": "test_component_2",
            "ResourceList": [
                "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i",
                "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
            ]
        }
    ],
    "LogPatternSets": [
        {
            "PatternSetName": "pattern_set_1",
            "LogPatterns": [
                {
                    "PatternName": "deadlock_pattern",
                    "Pattern": ".*\sDeadlocked\sSchedulers(([^\w].*)|($))",
                    "Rank": 1
                }
            ]
        },
        {
            "PatternSetName": "pattern_set_2",
            "LogPatterns": [
                {
                    "PatternName": "error_pattern",
                    "Pattern": ".*[\s\[]ERROR[\s\]].*",
                    "Rank": 1
                },
                {
                    "PatternName": "warning_pattern",
                    "Pattern": ".*[\s\[]WARN(ING)?[\s\]].*",
                    "Rank": 10
                }
            ]
        }
    ]
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
    ResourceGroupName: my_resource_group
    CWEEnabled: true
    OpsCenterEnabled: true
    OpsItemSNSTopicArn: arn:aws:sns:us-east-1:123456789012:my_topic
    AutoConfigurationEnabled: true
    Tags:
        - Key: key1
          Value: value1
        - Key: key2
          Value: ''
    CustomComponents:
        - ComponentName: test_component_1
          ResourceList:
              - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
        - ComponentName: test_component_2
          ResourceList:
              - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
              - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
    LogPatternSets:
        - PatternSetName: pattern_set_1
          LogPatterns:
```

```
- PatternName: deadlock_pattern
  Pattern: ".*\sDeadlocked\sSchedulers(([^\w].*)|($))"
  Rank: 1
- PatternSetName: pattern_set_2
  LogPatterns:
    - PatternName: error_pattern
      Pattern: ".*[\s\[]ERROR[\s\]].*"
      Rank: 1
    - PatternName: warning_pattern
      Pattern: ".*[\s\[]WARN(ING)?[\s\[]].*"
      Rank: 10
```

Create an Application Insights application with CUSTOM mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `CUSTOM`, which causes this component to be configured with the configuration specified in `CustomComponentConfiguration`. For more information, see [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
  "Type": "AWS::ApplicationInsights::Application",
  "Properties": {
    "ResourceGroupName": "my_resource_group",
    "ComponentMonitoringSettings": [
      {
        "ComponentARN": "my_component",
        "Tier": "SQL_SERVER",
        "ComponentConfigurationMode": "CUSTOM",
        "CustomComponentConfiguration": {
          "ConfigurationDetails": {
            "AlarmMetrics": [
              {
                "AlarmMetricName": "StatusCheckFailed"
              },
              ...
            ],
            "Logs": [
              {
                "LogGroupName": "my_log_group_1",
                "LogPath": "C:\LogFolder_1\*",
                "LogType": "DOT_NET_CORE",
                "Encoding": "utf-8",
                "PatternSet": "my_pattern_set_1"
              },
              ...
            ],
            "WindowsEvents": [
              {
                "LogGroupName": "my_windows_event_log_group_1",
                "EventName": "Application",
                "EventLevels": [
                  "ERROR",
                  "WARNING",
                  ...
                ]
              }
            ]
          }
        }
      }
    ]
  }
}
```

```
        ...
        ],
        "Encoding": "utf-8",
        "PatternSet": "my_pattern_set_2"
    },
    ...
],
"Alarms": [
    {
        "AlarmName": "my_alarm_name",
        "Severity": "HIGH"
    },
    ...
]
},
"SubComponentTypeConfigurations": [
{
    "SubComponentType": "EC2_INSTANCE",
    "SubComponentConfigurationDetails": {
        "AlarmMetrics": [
            {
                "AlarmMetricName": "DiskReadOps"
            },
            ...
        ],
        "Logs": [
            {
                "LogGroupName": "my_log_group_2",
                "LogPath": "C:\\\\LogFolder_2\\\\*",
                "LogType": "IIS",
                "Encoding": "utf-8",
                "PatternSet": "my_pattern_set_3"
            },
            ...
        ],
        "WindowsEvents": [
            {
                "LogGroupName": "my_windows_event_log_group_2",
                "EventName": "Application",
                "EventLevels": [
                    "ERROR",
                    "WARNING",
                    ...
                ],
                "Encoding": "utf-8",
                "PatternSet": "my_pattern_set_4"
            },
            ...
        ]
    }
}
]
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
    ResourceGroupName: my_resource_group
    ComponentMonitoringSettings:
```

```
- ComponentARN: my_component
Tier: SQL_SERVER
ComponentConfigurationMode: CUSTOM
CustomComponentConfiguration:
  ConfigurationDetails:
    AlarmMetrics:
      - AlarmMetricName: StatusCheckFailed
      ...
    Logs:
      - LogGroupName: my_log_group_1
        LogPath: C:\LogFolder_1\*
        LogType: DOT_NET_CORE
        Encoding: utf-8
        PatternSet: my_pattern_set_1
      ...
  WindowsEvents:
    - LogGroupName: my_windows_event_log_group_1
      EventName: Application
      EventLevels:
        - ERROR
        - WARNING
        ...
      Encoding: utf-8
      PatternSet: my_pattern_set_2
    ...
  Alarms:
    - AlarmName: my_alarm_name
      Severity: HIGH
      ...
SubComponentTypeConfigurations:
- SubComponentType: EC2_INSTANCE
  SubComponentConfigurationDetails:
    AlarmMetrics:
      - AlarmMetricName: DiskReadOps
      ...
    Logs:
      - LogGroupName: my_log_group_2
        LogPath: C:\LogFolder_2\*
        LogType: IIS
        Encoding: utf-8
        PatternSet: my_pattern_set_3
      ...
    WindowsEvents:
      - LogGroupName: my_windows_event_log_group_2
        EventName: Application
        EventLevels:
          - ERROR
          - WARNING
          ...
        Encoding: utf-8
        PatternSet: my_pattern_set_4
      ...
```

Create an Application Insights application with DEFAULT mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
 - Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT` and `Tier` to `SQL_SERVER`, which causes this component to be configured with the configuration settings

that Application Insights recommends for the `SQL_Server` tier. For more information, see [DescribeComponentConfiguration](#) and [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{  
    "Type": "AWS::ApplicationInsights::Application",  
    "Properties": {  
        "ResourceGroupName": "my_resource_group",  
        "ComponentMonitoringSettings": [  
            {  
                "ComponentARN": "my_component",  
                "Tier": "SQL_SERVER",  
                "ComponentConfigurationMode": "DEFAULT"  
            }  
        ]  
    }  
}
```

Template in YAML format

```
---  
Type: AWS::ApplicationInsights::Application  
Properties:  
  ResourceGroupName: my_resource_group  
  ComponentMonitoringSettings:  
    - ComponentARN: my_component  
      Tier: SQL_SERVER  
      ComponentConfigurationMode: DEFAULT
```

Create an Application Insights application with `DEFAULT_WITH_OVERWRITE` mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT_WITH_OVERWRITE` and `tier` to `DOT_NET_CORE`, which causes this component to be configured with the configuration settings that Application Insights recommends for the `DOT_NET_CORE` tier. Overwritten configuration settings are specified in the `DefaultOverwriteComponentConfiguration`:
 - At the component level `AlarmMetrics` settings are overwritten.
 - At the sub-component level, for the `EC2_Instance` type sub-components, `Logs` settings are overwritten.

For more information, see [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
```

Amazon CloudWatch User Guide
Create an Application Insights application
with `DEFAULT_WITH_OVERWRITE`
mode component configuration

```
"Type": "AWS::ApplicationInsights::Application",
"Properties": {
    "ResourceGroupName": "my_resource_group",
    "ComponentMonitoringSettings": [
        {
            "ComponentName": "my_component",
            "Tier": "DOT_NET_CORE",
            "ComponentConfigurationMode": "DEFAULT_WITH_OVERWRITE",
            "DefaultOverwriteComponentConfiguration": {
                "ConfigurationDetails": {
                    "AlarmMetrics": [
                        {
                            "AlarmMetricName": "StatusCheckFailed"
                        }
                    ]
                },
                "SubComponentTypeConfigurations": [
                    {
                        "SubComponentType": "EC2_INSTANCE",
                        "SubComponentConfigurationDetails": {
                            "Logs": [
                                {
                                    "LogGroupName": "my_log_group",
                                    "LogPath": "C:\\LogFolder\\*",
                                    "LogType": "IIS",
                                    "Encoding": "utf-8",
                                    "PatternSet": "my_pattern_set"
                                }
                            ]
                        }
                    }
                ]
            }
        }
    ]
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
    - ComponentName: my_component
      Tier: DOT_NET_CORE
      ComponentConfigurationMode: DEFAULT_WITH_OVERWRITE
      DefaultOverwriteComponentConfiguration:
        ConfigurationDetails:
          AlarmMetrics:
            - AlarmMetricName: StatusCheckFailed
        SubComponentTypeConfigurations:
          - SubComponentType: EC2_INSTANCE
            SubComponentConfigurationDetails:
              Logs:
                - LogGroupName: my_log_group
                  LogPath: C:\\LogFolder\\*
                  LogType: IIS
                  Encoding: utf-8
                  PatternSet: my_pattern_set
```

Tutorial: Set up monitoring for SAP HANA

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP HANA databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and facilitate mean time to resolution (MTTR) for your SAP HANA databases.

Application Insights for SAP HANA topics

- [Supported environments \(p. 677\)](#)
- [Supported operating systems \(p. 677\)](#)
- [Features \(p. 678\)](#)
- [Prerequisites \(p. 678\)](#)
- [Set up your SAP HANA database for monitoring \(p. 679\)](#)
- [Manage monitoring of your SAP HANA database \(p. 680\)](#)
- [Configure the alarm threshold \(p. 680\)](#)
- [View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights \(p. 681\)](#)
- [Anomaly detection for SAP HANA \(p. 683\)](#)
- [Troubleshooting Application Insights for SAP HANA \(p. 684\)](#)

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns. You provide and install SAP HANA database software and supported SAP application software.

- **SAP HANA database on a single Amazon EC2 instance** — SAP HANA in a single-node, scale-up architecture, with up to 24TB of memory.
- **SAP HANA database on multiple Amazon EC2 instances** — SAP HANA in a multi-node, scale-out architecture.
- **Cross-AZ SAP HANA database high availability setup** — SAP HANA with high availability configured across two Availability Zones using SUSE/RHEL clustering.

Supported operating systems

CloudWatch Application Insights for SAP HANA supports x86-64 architecture on the following operating systems:

- SUSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 For SAP
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- RedHat Linux 8.2 For SAP With High Availability and Update Services
- RedHat Linux 8.1 For SAP With High Availability and Update Services
- RedHat Linux 7.9 For SAP With High Availability and Update Services

Features

CloudWatch Application Insights for SAP HANA provides the following features:

- Automatic SAP HANA workload detection
- Automatic SAP HANA alarm creation based on static threshold
- Automatic SAP HANA alarm creation based on anomaly detection
- Automatic SAP HANA log pattern recognition
- Health dashboard for SAP HANA
- Problem dashboard for SAP HANA

Prerequisites

You must perform the following prerequisites to configure an SAP HANA database with CloudWatch Application Insights:

- **SAP HANA** — Install a running and reachable SAP HANA database 2.0 SPS05 on an Amazon EC2 instance.
- **SAP HANA database user** — Run the following SQL commands to create a user with monitoring roles.

```
su - <sid>adm
hdbsql -u SYSTEM -p <SYSTEMDB password> -d SYSTEMDB
CREATE USER CW_HANADB_EXPORTER_USER PASSWORD <Monitoring user password> NO
FORCE_FIRST_PASSWORD_CHANGE;
CREATE ROLE CW_HANADB_EXPORTER_ROLE;
GRANT MONITORING TO CW_HANADB_EXPORTER_ROLE;
GRANT CW_HANADB_EXPORTER_ROLE TO CW_HANADB_EXPORTER_USER;
```

- **Python 3.6** — Install Python 3.6 or later versions on your operating system. If Python is not detected on your operating system, Python 3.8 will be installed.
- **Pip3** — Install the installer program, pip3, on your operating system. If pip3 is not detected on your operating system, it will be installed.
- **Amazon CloudWatch agent** — Make sure that you are not running a preexisting Amazon CloudWatch agent on your Amazon EC2 instance.
- **AWS Systems Manager enablement** — Install SSM Agent on your instances, and the instances must be enabled for SSM. For information about how to install the SSM agent, see [Working with SSM Agent](#) in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** — You must attach the following Amazon EC2 instance roles to configure your database.
 - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see [AWS Systems Manager identity-based policy examples](#).
 - You must attach the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. For more information, see [Create IAM Roles and Users for Use With CloudWatch Agent](#).
 - You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in AWS Secrets Manager. For more information about inline policies, see [Inline policies](#) in the *AWS Identity and Access Management User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
```

```
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue"
        ],
        "Resource": "arn:aws:secretsmanager:*::secret:ApplicationInsights-*"
    }
}
```

- **AWS resource groups** — You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP HANA database.
- **IAM permissions** — For non-admin users, you must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see [IAM policy \(p. 616\)](#).
- **Service-linked role** — Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see [Using service-linked roles for CloudWatch Application Insights \(p. 930\)](#).

Set up your SAP HANA database for monitoring

To set up monitoring for your SAP HANA database, perform the following steps.

1. Navigate to the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group that contains your SAP HANA database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the [AWS Resource Groups User Guide](#).
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
6. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your AWS resources, provide an SNS topic ARN.
7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and AWS CloudFormation-based resource groups, with the exception of Application Auto Scaling groups. For more information, see [Tag Editor](#) in the [AWS Resource Groups and Tags User Guide](#).
8. Choose **Next** to continue to set up monitoring.
9. On the **Set up monitoring** page, the monitored components automatically detected by CloudWatch Application Insights are listed. Choose **Next**.
10. On the **Specify component details** page, enter the user name and password for your user.
11. Choose **Next**.
12. Review your application monitoring configuration, and choose **Submit**.

13. The application details page opens, where you can view the **Application summary**, and the list of **Monitored components** and **Unmonitored components**. If you select the radio button next to a component, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Manage monitoring of your SAP HANA database

You can manage user credentials, metrics, and log paths for your SAP HANA database by performing the following steps:

1. Navigate to the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
6. Under **Metrics**, you can choose the SAP HANA metrics according to your requirements. SAP HANA metric names are prefixed with hanadb. You can add up to 40 metrics per component.
7. Under **HANA configuration**, enter the password and user name for the SAP HANA database. This is the user name and password that Amazon CloudWatch agent uses to connect to the SAP HANA database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Configure the alarm threshold

CloudWatch Application Insights automatically creates a Amazon CloudWatch metric for the alarm to watch, along with the threshold for that metric. The alarm changes to the **ALARM** state when the metric surpasses the threshold for a specified number of evaluation periods. Note that these settings are not retained by Application Insights.

To edit an alarm for a single metric, perform the following steps:

1. Navigate to the [CloudWatch console](#).
2. In the left navigation pane, choose **Alarms>All alarms**.
3. Select the radio button next to the alarm that was automatically created by CloudWatch Application Insights. Then choose **Actions**, and select **Edit** from the dropdown menu.
4. Edit the following parameters under **Metric**.
 - a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile. For example, p95 . 45.
 - b. Under **Period**, choose the evaluations period for the alarm. When you evaluate the alarm, each period is aggregated into one data point.
5. Edit the following parameters under **Conditions**.
 - a. Choose whether the metric must be greater than, less than, or equal to the threshold.

- b. Specify the threshold value.
6. Under **Additional configuration** edit the following parameters.
 - a. Under **Datapoints to alarm**, specify the number of data points, or evaluation periods, that must be in the **ALARM** state to initiate the alarm. When the two values match, an alarm is created that enters **ALARM** state if the designated number of consecutive periods are exceeded. To create an m out of n alarm, specify a lower value for the first data point than for the second. For more information about evaluating alarms, see [Evaluating an alarm..](#)
 - b. Under **Missing data treatment**, choose the behavior of the alarm when some data points are missing. For more information about missing data treatment, see [Configuring how CloudWatch alarms treat missing data](#).
 - c. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information about percentiles with low samples, see [Percentile-based CloudWatch alarms and low data samples \(p. 136\)](#) .
7. Choose **Next**.
8. Under **Notification**, select an SNS topic to notify when the alarm is in **ALARM** state, **OK** state, or **INSUFFICIENT_DATA** state.
9. Choose **Update alarm**.

View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP HANA on Application Insights.

Troubleshooting topics

- [SAP HANA database reaches memory allocation limit \(p. 681\)](#)
- [Disk full event \(p. 682\)](#)
- [SAP HANA backup stopped running \(p. 683\)](#)

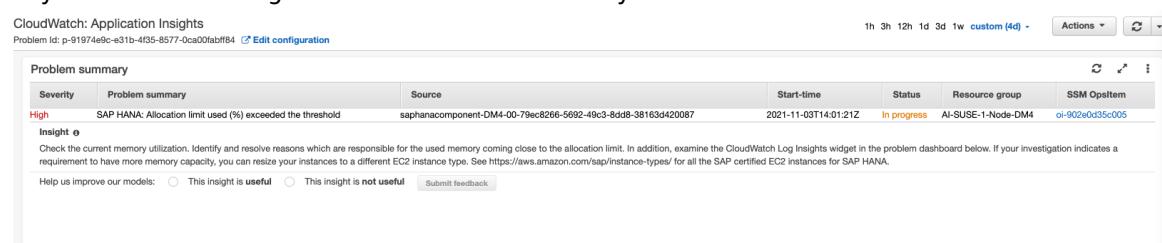
SAP HANA database reaches memory allocation limit

Description

Your SAP application that is backed by an SAP HANA database malfunctions because of high memory pressure, leading to application performance degradation.

Resolution

You can identify the application layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be because of a large data load in the SAP HANA system.

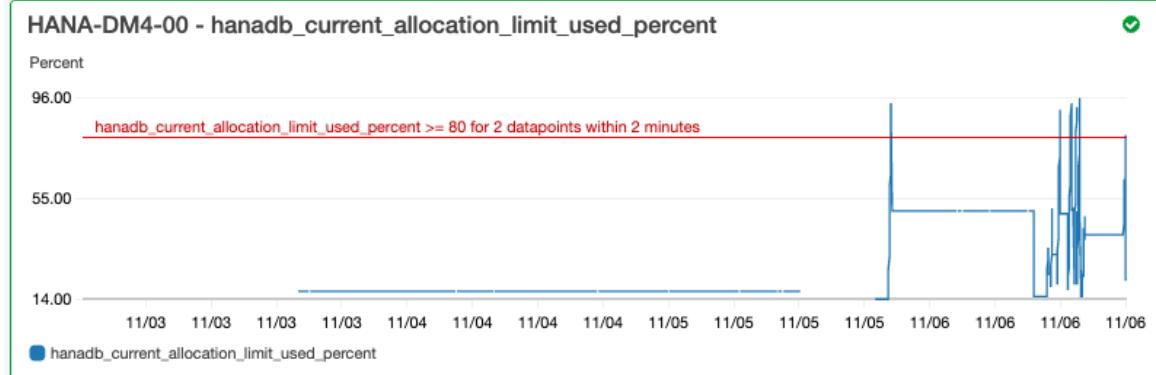


The screenshot shows the CloudWatch Application Insights interface with the following details:

- Problem summary:** SAP HANA: Allocation limit used (%) exceeded the threshold
- Source:** saphanacomponent-DM-00-79ec8266-5692-49c3-8dd8-38163d420087
- Start-time:** 2021-11-03T14:01:21Z
- Status:** In progress
- Resource group:** AI-SUSE-1-Node-DM4
- SSM OpsItem:** oi-902e0d35c005
- Severity:** High
- Insight:** SAP HANA: Allocation limit used (%) exceeded the threshold
- Message:** Check the current memory utilization. Identify and resolve reasons which are responsible for the used memory coming close to the allocation limit. In addition, examine the CloudWatch Log Insights widget in the problem dashboard below. If your investigation indicates a requirement to have more memory capacity, you can resize your instances to a different EC2 instance type. See <https://aws.amazon.com/sap/instance-types/> for all the SAP certified EC2 instances for SAP HANA.
- Feedback:** Help us improve our models: This insight is useful This insight is not useful

The used memory allocation exceeds the threshold of 80 percent of the total memory allocation limit.

EC2 instance group - HANA-DM4-00



The log group shows the scheme BNR–DATA and table IMDBMASTER_30003 ran out of memory. In addition, the log group shows the exact time of the issue, current global location limit, shared memory, code size, and OOM reservation allocation size.

```
Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM4, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.OutOfMemory

# :@timestamp :@message
1 2021-11-06T13:31:23.317Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
[2467]([311269])22:0631545 2021-11-06 13:00:44.099578 e OOM_Notification_Statement.cc(45430) : com exception occurred at :/indbmaster:30003: conn_id=311269, stmt_hash=17elcccbb5f460604ce08c9869fd81, sql=CAL...
2 2021-11-06T13:31:23.316Z [3833]([311513])22:067162 2021-11-06 13:31:17.163649 e Memory mReportMemoryProblems.cpp(10885) : OUT OF MEMORY occurred.
3 2021-11-06T13:31:23.316Z Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
4 2021-11-06T13:31:23.316Z [2422]([311269])22:067162 Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
5 2021-11-06T13:31:23.316Z [2422]([311269])22:067162 Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
6 2021-11-06T13:31:23.316Z [2422]([311269])22:067162 Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
7 2021-11-06T13:31:23.316Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
8 2021-11-06T13:31:17.318Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
9 2021-11-06T13:31:17.318Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
10 2021-11-06T13:31:17.317Z [3033]([311513])22:067162 2021-11-06 13:31:17.109322 e Memory mReportMemoryProblems.cpp(1032) : OUT OF MEMORY occurred. For Pool/malloc/l1hdbbaseMem, size=422808, alignment=8, Flags 0x0, reason GLOBAL_ALLOC...
11 2021-11-06T13:31:17.317Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
12 2021-11-06T13:31:17.317Z [3833]([311513])22:067162 2021-11-06 13:31:17.163649 e Memory mReportMemoryProblems.cpp(10885) : OUT OF MEMORY occurred.
13 2021-11-06T13:31:17.317Z Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
14 2021-11-06T13:31:17.317Z [2422]([311269])22:067162 Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
15 2021-11-06T13:31:17.317Z [2422]([311269])22:067162 Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(Unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
16 2021-11-06T13:31:17.317Z [2422]([311269])22:067162 Current callstack: 1: 0x00007f8245386d35 In MemoryManager::PoolAllocатор::notifyOOMmap(UnSigned long, unsigned long, bool, ltt::allocation_failure_type, bool)*>x1b1 at mPoolAllocатор.cpp:2284 ([libhdbbasis.so] 2: 0x00007f824524a70d ...
17 2021-11-06T13:31:17.317Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
18 2021-11-06T13:31:16.317Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
19 2021-11-06T13:31:16.317Z GLOBAL_ALLOCATION_LIMIT (CAL) = 55.78gb (39981081728b), SHARED_MEMORY = 567.77mb (395357696b), CODE_SIZE = 2.94gb (316259872b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (108810752b)
```

Disk full event

Description

Your SAP application that is backed by an SAP HANA database stops responding, which leads to an inability to access the database.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be that the administrator failed to enable automatic log backup, which caused the sap/hana/log directory to fill up.

Problem summary							
Severity	Problem summary	Source	Start-time	Status	Resource group	SSM Opstern	
Medium	SAP HANA: DISK FULL error has been detected	i-049851dc9a2ab15cc	2021-11-05T18:07:29Z	In progress	AI-SUSE-1-Node-DM-02	oi-8bf4cb8fcff8	
Insight							
If the HANA database does not accept any of the new requests due to log volume is full. We strongly advise against remove either files or log files using operating system tools as this will corrupt the database. The recommendation is to follow SAP Note 1679938 to temporarily free up space in the log volume, this way you should be able to start up the database for root cause analysis and problem resolution.							
Help us improve our models: <input type="radio"/> This insight is useful <input type="radio"/> This insight is not useful Submit feedback							

The log group widget in the problem dashboard shows the DISKFULL event.

Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM2, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.DiskFull

```
# :@timestamp :@message
▼ 1 2021-11-06T18:00:20.072Z [26768]{-1}[-1] 2021-11-06 18:00:16.556583 i EventHandler LocalFileCallback.cpp(00517) : [DISKFULL] restarting queue with 1 requests
@ingestionTime 1636221622489
@log [REDACTED]:SAP_HANA_TRACE-AI-SUSE-1-Node-DM2
@logStream i-[REDACTED]
@message [26768]{-1}[-1] 2021-11-06 18:00:16.556583 i EventHandler LocalFileCallback.cpp(00517) : [DISKFULL] restarting queue with 1 requests
@timestamp 1636221620072
```

SAP HANA backup stopped running

Description

Your SAP application that is backed by an SAP HANA database has stopped working.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets.

The log group widget in the problem dashboard shows the `ACCESS DENIED` event. This includes additional information, such as the S3 bucket, the S3 bucket folder, and the S3 bucket Region.

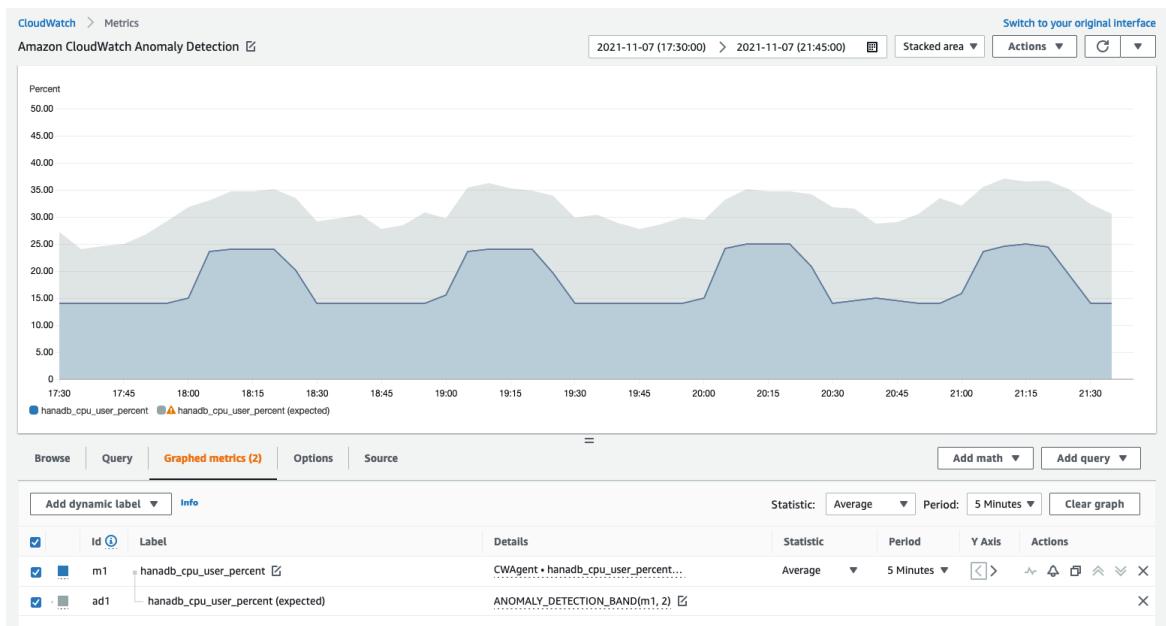
Log Group: SAP_HANA_LOGS-AI-SUSE-1-Node-DM3, Log Type: SAP_HANA_LOGS, AWS::SAPHANA.BackupErrorAccessDenied

```
# :@timestamp :@message
▼ 1 2021-11-06T20:28:34.502Z 2021-11-06 20:28:34.493 backint terminated: pid: 21196 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
@ingestionTime 1636230519523
@log 743491381166:SAP_HANA_LOGS-AI-SUSE-1-Node-DM3
@logStream i-00100000025F52321b
@message 2021-11-06 20:28:34.493 backint terminated:
pid: 21196
exit code: 1
output:
exception:
exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243)
Backint exited with exit code 1 instead of 0. console output: time=2021-11-06T20:28:34Z level=info msg="Starting execution." time="2021-11-06T20:28:34Z" level=info msg="Loading configuration file /usr/sap/DM3/SYS/global/hdb/opt/hdbconfig
@timestamp 1636230519496
▶ 2 2021-11-06T20:28:34.502Z 2021-11-06 20:28:34.493 backint terminated: pid: 21196 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
▶ 3 2021-11-06T20:27:22.894Z 2021-11-06 20:27:22.999 backint terminated: pid: 21089 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
▶ 4 2021-11-06T20:26:46.052Z 2021-11-06 20:26:41.277 backint terminated: pid: 208947 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
▶ 5 2021-11-06T20:26:39.052Z 2021-11-06 20:26:34.218 backint terminated: pid: 208931 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
▶ 6 2021-11-06T20:26:22.949Z 2021-11-06 20:26:22.823 backint terminated: pid: 208876 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
▶ 7 2021-11-06T20:25:41.183Z 2021-11-06 20:25:41.136 backint terminated: pid: 208814 exit code: 1 output: exception: exception 1: no.110587 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console ...
```

Anomaly detection for SAP HANA

For specific SAP HANA metrics, such as the number of thread count, CloudWatch applies statistical and machine learning algorithms to define the threshold. These algorithms continuously analyze the metrics of the SAP HANA database, determine normal baselines, and surface anomalies with minimal user intervention. The algorithms generate an anomaly detection model, which generates a range of expected values that represent normal metric behavior.

Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes can be hourly, daily, or weekly, as shown in the following examples of the SAP HANA CPU usage.



After you create a model, CloudWatch anomaly detection continuously evaluates the model and makes adjustments to it to ensure that it is as accurate as possible. This includes retraining the model to adjust if the metric values evolve over time or experience sudden changes. It also includes predictors to improve the models for metrics that are seasonal, spiky, or sparse.

Troubleshooting Application Insights for SAP HANA

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitor metrics

Error returned: Component cannot have more than 60 monitored metrics.

Root cause: The current metric limit is 60 monitor metrics per component.

Resolution: Remove metrics that are not necessary to adhere to the limit.

No SAP metrics or alarms appear after the onboarding process.

Error returned: In AWS Systems Manager Run command, the AWS-ConfigureAWS Package failed.

The output shows the following error:

```
Unable to find a host with system database, for more info rerun using -v
```

Step 2 - Command description and status		
Status ✖ Failed	Detailed status ✖ Failed	Response code 1
Step name configurePackage	Start time Tue, 09 Nov 2021 10:00:47 GMT	Finish time Tue, 09 Nov 2021 10:00:59 GMT
▶ Output		
▼ Error		
<p>The command output displays a maximum of 48,000 characters. You can view the complete command output in either Amazon S3 or CloudWatch Logs, if you specify an S3 bucket or a logs group when you run the command.</p> <pre>password=db_creds["password"]) File "./install_utils.py", line 102, in get_master_host raise InstallationError("Unable to find a host with system database, for more info rerun using -v") __main__.InstallationError: Unable to find a host with system database, for more info rerun using -v failed to run commands: exit status 1 Failed to install package; install status Failed</pre> <div style="text-align: right;">Copy Download</div>		

Resolution: The user name and password may be incorrect. Verify that the user name and password are valid, and rerun the onboarding process.

Tutorial: Set up monitoring for .NET applications using SQL Server

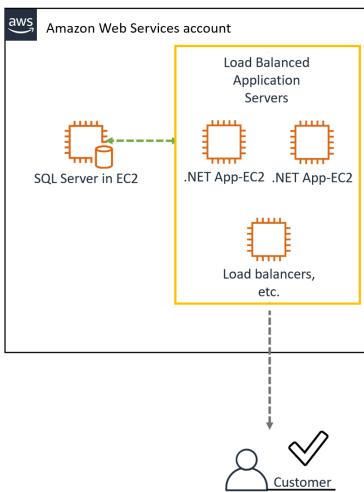
This tutorial demonstrates how to configure CloudWatch Application Insights to monitor an example solution and then simulate problem scenarios to test the solution. In this example, a load balanced web application using SQL Server on the backend is deployed. The web application and SQL Server are hosted on separate EC2 instances.

Tutorial components

- [Use case scenario \(p. 685\)](#)
- [Prerequisites \(p. 686\)](#)
- [Deploy resources for example scenario \(p. 686\)](#)
- [Set up monitoring with Amazon CloudWatch Application Insights \(p. 687\)](#)
- [Simulate problem scenarios and view insights \(p. 688\)](#)

Use case scenario

In this scenario, a .NET application using SQL Server on the backend runs on an Amazon EC2 instance. The deployment is configured with two load-balanced EC2 instances hosting the Barley Adventure Works application. Both instances access SQL Server, which is hosted on a separate EC2 instance. Monitors are set up with CloudWatch Application Insights to quickly identify, isolate, and resolve application issues.



Prerequisites

To complete the steps in this tutorial, you must have an AWS account.

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

To sign up for an AWS account

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

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

Deploy resources for example scenario

A CloudFormation template is provided to automate the deployment scenario for testing. The template deploys the following instances:

- An EC2 instance that hosts the Microsoft SQL Server database.
- Two load-balanced EC2 instances. Each load-balanced instance hosts the Barley Adventure Works web application.

The template deployment takes approximately 10 minutes to complete.

Steps to deploy the CloudFormation stack

1. Choose **Create Stack>With new resources (standard)** from the AWS CloudFormation landing page at <https://console.aws.amazon.com/cloudformation> to launch a CloudFormation stack in your account in the US East (N. Virginia) Region. This template is available only in the `us-east-1` Region.
2. Select **Template is ready** on the **Create Stack** page.
3. Under **Specify template**, select **Amazon S3 URL** and enter the following S3 URL path: <https://application-insights-demo-resources.s3.amazonaws.com/SampleApp.yml>. Choose **Next**.

Note

This CloudFormation template can also be found in the aws-samples GitHub repo at the following location: <https://github.com/aws-samples/application-insights-sample-application/blob/master/SampleApp.yml>.

4. On the **Specify stack details** page, enter a name for the stack, such as `ApplicationInsightsTest`.
5. Review the default parameters under **Parameters** and modify the values to your preferences. Enter a password for SQLServer. Verify that you follow the SQL Server password complexity requirements or the application will not work as expected. Enter an existing EC2 key pair or create a new one in the EC2 console. Choose **Next**.
6. On the **Configure stack options** page, under **Tags**, optionally add tags to help you identify your stack. Select **Next**.
7. Review and confirm the settings on the **Review** page. Select the box acknowledging that the template may create AWS Identity and Access Management (IAM) resources.
8. Choose **Create stack** to deploy the stack. Reminder that you can only deploy this template from the `us-east-1` Region.
9. Monitor the status of the stack deployment from the **Events** tab of the Cloud Formation stack page. When the stack is successfully deployed, continue to the next section.

Set up monitoring with Amazon CloudWatch Application Insights

This section demonstrates how to create a resource group from the resources deployed by the CloudFormation template, and how to add the resource group to CloudWatch Application Insights for monitoring.

Create resource group

1. Navigate to the [AWS Resource Groups console](#) and choose **Create resource group**.
2. On the **Create query-based group** page, under **Group type**, select **CloudFormation stack based**.
3. Under **Grouping criteria**, select the CloudFormation stack you created in the previous section (`ApplicationInsightsTest`) from the dropdown list. Keep resource types as **All supported resource types**.
4. Under **Group details**, enter a **Group name**, such as `application-insights-resource-group`, and an optional description of the resource group. Then, choose **Create group**.

Set up resource group monitoring on CloudWatch Application Insights

1. Navigate to the [Amazon CloudWatch console](#) and choose **Application Insights** under **Insights** on the left navigation pane.
2. Choose **View applications** next to **Application Insights**.
3. On the **Applications monitored** page, choose **Add an application**.
4. Under **Resource Group selection**, select the resource group you created in the previous procedure (`application-insights-resource-group`) and choose **Add application**.
5. On the **Overview** page, refresh your browser to display the application components in your resource group.
6. Select the **Application Load Balancer** group and choose **Manage monitoring**.
7. On the **Manage monitoring** page, select **Enable monitoring**.

8. Choose **Save**.
9. To enable monitoring for the SQL Server instance, select the SQL Server EC2 instance in the **Application components** section and repeat the previous steps for enabling monitoring from the **Manage monitoring** page.

The following metrics are monitored for the SQL Server instance:

- CPUUtilization
- StatusCheckFailed
- Memory % Committed Bytes in Use
- Memory Available Mbytes
- Network Interface Bytes Total/sec
- Paging File % Usage
- Physical Disk % Disk Time
- Processor % Processor Time
- SQLServer:Buffer Manager cache hit ratio
- SQLServer:Buffer Manager life expectancy
- SQLServer:General Statistics Processes blocked
- SQLServer:General Statistics User Connections
- SQLServer:Locks Number of Deadlocks/sec
- SQLServer:SQL Statistics Batch Requests/sec
- System Processor Queue Length

The following metrics are monitored for the volumes attached to the SQL Server instance:

- VolumeReadBytes
- VolumeWriteBytes
- VolumeReadOps
- VolumeWriteOps
- VolumeTotalReadTime
- VolumeTotalWriteTime
- VolumIdleTime
- VolumeQueueLength
- VolumeThroughputPercentage
- VolumeConsumedReadWriteOps
- BurstBalance

10. When monitoring is enabled for both the load balancer and the SQL Server instance, the resource group to which they belong displays a status of **Enabled** on the resource group **Overview** page.

Simulate problem scenarios and view insights

This section describes how to create a SQL login failure, a SQL memory pressure event, and an HTTP 500 error so that you can view the error details on the CloudWatch Application Insights dashboard.

Problems

- [Simulate SQL login failure \(p. 689\)](#)
- [Simulate high memory pressure \(p. 689\)](#)
- [Simulate an HTTP 500 error \(p. 689\)](#)

Simulate SQL login failure

To simulate a SQL login failure and view the problem from the CloudWatch dashboard, perform the following steps.

1. Log in to the EC2 instance provisioned for your SQL Server instance (M4 instance type) using the key pair you selected when you created the CloudFormation stack.
2. From the **Start** menu, launch SQL Management Studio.
3. Enter a username and an incorrect password and choose **Connect**. A message appears indicating that the login has failed. Repeat this step a few more times.
4. From the CloudWatch Application Insights **Problems detected** page (at the bottom of the [CloudWatch console landing page](#)), the error should appear under the problem summary as **SQL: Login Failure**. To see more details about the problem, select the problem link.

Simulate high memory pressure

To simulate a high memory pressure event, which can lead to application performance degradation and timeout errors on the web servers and load balancers, perform the following steps.

1. Launch SQL Management Studio from the SQL server instance and login using the Windows Administrator account.
2. Right-click on the database server, choose **Properties**, and select **Memory**.
3. Under the **Server memory options**, reduce the **Maximum server memory** to 256 KB.
4. In a new query window, run the following SQL query:

```
SELECT COUNT(*)
FROM [AdventureWorks2016].[Sales].[Customer]
```

A message appears indicating that there is insufficient memory to run the query.

5. From the CloudWatch Application Insights **Problems detected** page (at the bottom of the [CloudWatch console landing page](#)), the error should appear under the problem summary as **SQL: Memory Pressure**. To see more details about the problem, select the problem link.

Simulate an HTTP 500 error

HTTP requests for an unhandled HTTP request to a web application results in an HTTP 500 error. To simulate an HTTP 500 error, perform the following steps.

1. From the AWS Management Console, navigate to the AWS CloudFormation console.
2. Choose the **Outputs** tab of the **AppInsightsTest** CloudFormation stack that you previously launched.
3. Open the URL displayed under **Value** for the AdventureWorks application in a web browser.
4. Navigate to the customer details page of the Barley Adventure Works web application by suffixing the previously mentioned URL with `sampleapp/SalesOrderDetails/edit/5`.
5. Refresh the compiled URL request several times. Your URL should look something like this: `http://<YourURL>.us-east-1.elb.amazonaws.com/sampleapp/SalesOrderDetails/edit/5`.

An error message appears indicating that the file or directory cannot be found.

6. From the CloudWatch Application Insights **Problems detected** page (at the bottom of the [CloudWatch console landing page](#)), the error should appear under the problem summary as **ALB: Backend 5XX errors**. To see more details about the problem, select the problem link.

View and troubleshoot problems detected by Amazon CloudWatch Application Insights

The topics in this section provide detailed information about the detected problems and insights displayed by Application Insights. It also provides suggested resolutions for detected issues with your account or your configuration.

Troubleshooting topics

- [CloudWatch console overview \(p. 690\)](#)
- [Application Insights problem summary page \(p. 690\)](#)
- [Feedback \(p. 691\)](#)
- [Configuration errors \(p. 691\)](#)

CloudWatch console overview

An overview of problems that impact your monitored applications can be found under the CloudWatch Application Insights pane in the overview page of the [CloudWatch console](#). For more information, see [Get started with Amazon CloudWatch Application Insights \(p. 615\)](#).

The CloudWatch Application Insights overview pane displays the following:

- The severity of the problems detected: High/Medium/Low
- A short summary of the problem
- The problem source
- The time the problem started
- The resolution status of the problem
- The affected resource group

To view the details of a specific problem, under **Problem Summary**, select the description of the problem. A detailed dashboard displays insights into the problem and related metric anomalies and snippets of log errors. You can provide feedback on the relevance of the insight by selecting whether it is useful.

If a new resource is detected that is not configured, the problem summary description takes you to the **Edit configuration** wizard to configure your new resource. You can view or edit your resource group configuration by choosing **View/edit configuration** in the upper right-hand corner of the detailed dashboard.

To return to the overview, choose **Back to overview**, which is next to the CloudWatch Application Insights detailed dashboard header.

Application Insights problem summary page

Application Insights problem summary page

CloudWatch Application Insights provides the following information about detected problems on the problem summary page:

- A short summary of the problem
- The start time and date of the problem
- The problem severity: High/Medium/Low

- The status of the detected problem: In-progress/Resolved
- Insights: Automatically generated insights on the detected problem and possible root cause
- Feedback on insights: Feedback you have provided about the usefulness of the insights generated by CloudWatch Application Insights
- Related observations: A detailed view of the metric anomalies and error snippets of relevant logs related to the problem across various application components

Feedback

Feedback

You can provide feedback on the automatically generated insights on detected problems by designating them useful or not useful. Your feedback on the insights, along with your application diagnostics (metric anomalies and log exceptions), are used to improve the future detection of similar problems.

Configuration errors

CloudWatch Application Insights uses your configuration to create monitoring telemetries for the components. When Application Insights detects an issue with your account or your configuration, information is provided in the **Remarks** field of the Application summary about how to resolve the configuration issue for your application.

The following table shows suggested resolutions for specific remarks.

Remarks	Suggested resolution	Additional notes
The quota for alarms has already been reached.	By default, each AWS account can have 5,000 CloudWatch alarms per AWS Region. See CloudWatch Limits . When throttled by this limit, CloudWatch Application Insights cannot create all of the required alarms to monitor your application. To resolve this, raise the account limit for CloudWatch alarms.	n/a
The quota for CloudFormation has already been reached.	Application Insights creates one CloudFormation stack for each application to manage CloudWatch agent installation and configuration for all application components. By default, each AWS account can have 200 stacks. See AWS CloudFormation Limits . To resolve this, raise the limit for CloudFormation stacks.	n/a
No SSM instance role on the following instances.	For Application Insights to be able to install and configure CloudWatch agent on application instances, AmazonSSMManagedInstanceCore with SSM permission. After	Application Insights calls the SSM DescribeInstanceInformation API to get the list of instances

Remarks	Suggested resolution	Additional notes
	and CloudWatchAgentServerPolicy policies must be attached to the instance role.	the role is attached to the instance, it takes time for SSM to include the instance in the DescribeInstanceInformation result. Until SSM includes the instance in the result, NO_SSM_INSTANCE_ROLE error remains present for the application.
New components may need configuration.	Application Insights detects that there are new components in the application Resource Group. To resolve this, configure the new components accordingly.	n/a

Logs and metrics supported by Amazon CloudWatch Application Insights

The following lists show the supported logs and metrics for Amazon CloudWatch Application Insights.

CloudWatch Application Insights supports the following logs:

- Microsoft Internet Information Services (IIS) logs
- Error log for SQL Server on EC2
- Custom .NET application logs, such as Log4Net
- Windows Event logs, including Windows logs (System, Application, and Security) and Applications and Services log
- Amazon CloudWatch Logs for AWS Lambda
- Error log and slow log for RDS MySQL, Aurora MySQL, and MySQL on EC2
- Postgresql log for PostgreSQL RDS and PostgreSQL on EC2
- Amazon CloudWatch Logs for AWS Step Functions
- Execution logs and access logs (JSON, CSV, and XML, but not CLF) for API Gateway REST API stages
- Prometheus JMX exporter logs (EMF)
- Alert logs and listener logs for Oracle on Amazon RDS and Oracle on Amazon EC2
- Container logs routing from Amazon ECS containers to CloudWatch using [awslogs log driver](#).
- Container logs routing from Amazon ECS containers to CloudWatch using [FireLens container log router](#).
- Container logs routing from Amazon EKS or Kubernetes running on Amazon EC2 to CloudWatch using [Fluent Bit or Fluentd log processor](#) with Container Insights.
- SAP HANA trace and error logs
- HA Pacemaker logs

CloudWatch Application Insights supports metrics for the following application components:

- [Amazon Elastic Compute Cloud \(EC2\) \(p. 694\)](#)
 - [CloudWatch built-in metrics \(p. 694\)](#)
 - [CloudWatch agent metrics \(Windows server\) \(p. 695\)](#)

- [CloudWatch agent metrics \(Linux server\) \(p. 698\)](#)
- [Elastic Block Store \(EBS\) \(p. 700\)](#)
- [Elastic Load Balancer \(ELB\) \(p. 701\)](#)
- [Application ELB \(p. 701\)](#)
- [Amazon EC2 Auto Scaling groups \(p. 701\)](#)
- [Amazon Simple Queue Server \(SQS\) \(p. 702\)](#)
- [Amazon Relational Database Service \(RDS\) \(p. 703\)](#)
 - [RDS Database instances \(p. 703\)](#)
 - [RDS Database clusters \(p. 703\)](#)
- [AWS Lambda function \(p. 705\)](#)
- [Amazon DynamoDB table \(p. 705\)](#)
- [Amazon S3 bucket \(p. 705\)](#)
- [AWS Step Functions \(p. 706\)](#)
 - [Execution-level \(p. 706\)](#)
 - [Activity \(p. 706\)](#)
 - [Lambda function \(p. 707\)](#)
 - [Service integration \(p. 707\)](#)
 - [Step Functions API \(p. 707\)](#)
- [API Gateway REST API stages \(p. 708\)](#)
- [SAP HANA \(p. 708\)](#)
- [HA Cluster \(p. 712\)](#)
- [Java \(p. 713\)](#)
- [Amazon Elastic Container Service \(Amazon ECS\) \(p. 713\)](#)
 - [CloudWatch built-in metrics \(p. 714\)](#)
 - [Container Insights metrics \(p. 714\)](#)
 - [Container Insights Prometheus metrics \(p. 715\)](#)
- [Kubernetes on AWS \(p. 715\)](#)
 - [Container Insights metrics \(p. 716\)](#)
 - [Container Insights Prometheus metrics \(p. 716\)](#)
- [Amazon FSx \(p. 717\)](#)
- [Metrics with data points requirements \(p. 717\)](#)
 - [AWS/ApplicationELB \(p. 718\)](#)
 - [AWS/AutoScaling \(p. 718\)](#)
 - [AWS/EC2 \(p. 719\)](#)
 - [Elastic Block Store \(EBS\) \(p. 719\)](#)
 - [AWS/ELB \(p. 720\)](#)
 - [AWS/RDS \(p. 720\)](#)
 - [AWS/Lambda \(p. 721\)](#)
 - [AWS/SQS \(p. 722\)](#)
 - [AWS/CWAgent \(p. 722\)](#)
 - [AWS/DynamoDB \(p. 723\)](#)
 - [AWS/S3 \(p. 724\)](#)
 - [AWS/States \(p. 724\)](#)
 - [AWS/ApiGateway \(p. 725\)](#)
 - [AWS/SNS \(p. 725\)](#)

- [Recommended metrics \(p. 725\)](#)
- [Performance Counter metrics \(p. 740\)](#)

Amazon Elastic Compute Cloud (EC2)

CloudWatch Application Insights supports the following metrics:

Metrics

- [CloudWatch built-in metrics \(p. 694\)](#)
- [CloudWatch agent metrics \(Windows server\) \(p. 695\)](#)
- [CloudWatch agent metrics \(Linux server\) \(p. 698\)](#)

CloudWatch built-in metrics

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

CloudWatch agent metrics (Windows server)

.NET CLR Exceptions # of Exceps Thrown
.NET CLR Exceptions # of Exceps Thrown/Sec
.NET CLR Exceptions # of Filters/sec
.NET CLR Exceptions # of Finallys/sec
.NET CLR Exceptions Throw to Catch Depth/sec
.NET CLR Interop # of CCWs
.NET CLR Interop # of Stubs
.NET CLR Interop # of TLB exports/sec
.NET CLR Interop # of TLB imports/sec
.NET CLR Interop # of marshaling
.NET CLR Jit % Time in Jit
.NET CLR Jit Standard Jit Failures
.NET CLR Loading % Time Loading
.NET CLR Loading Rate of Load Failures
.NET CLR LocksAndThreads Contention Rate/sec
.NET CLR LocksAndThreads Queue Length/sec
.NET CLR Memory # Total Committed Bytes
.NET CLR Memory % Time in GC
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time
.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/sec
.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/sec
.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/sec
APP_POOL_WAS Total Worker Process Ping Failures
ASP.NET Application Restarts
ASP.NET Applications % Managed Processor Time (estimated)
ASP.NET Applications Errors Total/Sec
ASP.NET Applications Errors Unhandled During Execution/sec
ASP.NET Applications Requests in Application Queue
ASP.NET Applications Requests/Sec
ASP.NET Request Wait Time

ASP.NET Requests Queued
HTTP Service Request Queues CurrentQueueSize
LogicalDisk % Free Space
Memory % Committed Bytes In Use
Memory Available Mbytes
Memory Pages/sec
Network Interface Bytes Total/sec
Paging File % Usage
PhysicalDisk % Disk Time
PhysicalDisk Avg. Disk Queue Length
PhysicalDisk Avg. Disk sec/Read
PhysicalDisk Avg. Disk sec/Write
PhysicalDisk Disk Read Bytes/sec
PhysicalDisk Disk Reads/sec
PhysicalDisk Disk Write Bytes/sec
PhysicalDisk Disk Writes/sec
Processor % Idle Time
Processor % Interrupt Time
Processor % Processor Time
Processor % User Time
SQLServer:Access Methods Forwarded Records/sec
SQLServer:Access Methods Full Scans/sec
SQLServer:Access Methods Page Splits/sec
SQLServer:Buffer Manager Buffer cache hit ratio
SQLServer:Buffer Manager Page life expectancy
SQLServer:General Statistics Processes blocked
SQLServer:General Statistics User Connections
SQLServer:Latches Average Latch Wait Time (ms)
SQLServer:Locks Average Wait Time (ms)
SQLServer:Locks Lock Timeouts/sec
SQLServer:Locks Lock Waits/sec

SQLServer:Locks Number of Deadlocks/sec
SQLServer:Memory Manager Memory Grants Pending
SQLServer:SQL Statistics Batch Requests/sec
SQLServer:SQL Statistics SQL Compilations/sec
SQLServer:SQL Statistics SQL Re-Compilations/sec
System Processor Queue Length
TCPv4 Connections Established
TCPv6 Connections Established
W3SVC_W3WP File Cache Flushes
W3SVC_W3WP File Cache Misses
W3SVC_W3WP Requests/Sec
W3SVC_W3WP URI Cache Flushes
W3SVC_W3WP URI Cache Misses
Web Service Bytes Received/Sec
Web Service Bytes Sent/Sec
Web Service Connection attempts/sec
Web Service Current Connections
Web Service Get Requests/sec
Web Service Post Requests/sec
Bytes Received/sec
Normal Messages Queue Length/sec
Urgent Message Queue Length/sec
Reconnect Count
Unacknowledged Message Queue Length/sec
Messages Outstanding
Messages Sent/sec
Database Update Messages/sec
Update Messages/sec
Flushes/sec
Crypto Checkpoints Saved/sec
Crypto Checkpoints Restored/sec
Registry Checkpoints Restored/sec

Registry Checkpoints Saved/sec

Cluster API Calls/sec

Resource API Calls/sec

Cluster Handles/sec

Resource Handles/sec

CloudWatch agent metrics (Linux server)

cpu_time_active

cpu_time_guest

cpu_time_guest_nice

cpu_time_idle

cpu_time_iowait

cpu_time_irq

cpu_time_nice

cpu_time_softirq

cpu_time_steal

cpu_time_system

cpu_time_user

cpu_usage_active

cpu_usage_guest

cpu_usage_guest_nice

cpu_usage_idle

cpu_usage_iowait

cpu_usage_irq

cpu_usage_nice

cpu_usage_softirq

cpu_usage_steal

cpu_usage_system

cpu_usage_user

disk_free

disk_inodes_free

disk_inodes_used

disk_used
disk_used_percent
diskio_io_time
diskio_iops_in_progress
diskio_read_bytes
diskio_read_time
diskio_reads
diskio_write_bytes
diskio_write_time
diskio_writes
mem_active
mem_available
mem_available_percent
mem_buffered
mem_cached
mem_free
mem_inactive
mem_used
mem_used_percent
net_bytes_recv
net_bytes_sent
net_drop_in
net_drop_out
net_err_in
net_err_out
net_packets_recv
net_packets_sent
netstat_tcp_close
netstat_tcp_close_wait
netstat_tcp_closing
netstat_tcp_established
netstat_tcp_fin_wait1

netstat_tcp_fin_wait2
netstat_tcp_last_ack
netstat_tcp_listen
netstat_tcp_none
netstat_tcp_syn_recv
netstat_tcp_syn_sent
netstat_tcp_time_wait
netstat_udp_socket
processes_blocked
processes_dead
processes_idle
processes.paging
processes_running
processes_sleeping
processes_stopped
processes_total
processes_total_threads
processes_wait
processes_zombies
swap_free
swap_used
swap_used_percent

Elastic Block Store (EBS)

CloudWatch Application Insights supports the following metrics:

VolumeReadBytes
VolumeWriteBytes
VolumeReadOps
VolumeWriteOps
VolumeTotalReadTime
VolumeTotalWriteTime
VolumeIdleTime

VolumeQueueLength
VolumeThroughputPercentage
VolumeConsumedReadWriteOps
BurstBalance

Elastic Load Balancer (ELB)

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount
EstimatedALBConsumedLCUs
EstimatedALBNewConnectionCount
EstimatedProcessedBytes
HTTPCode_Backend_4XX
HTTPCode_Backend_5XX
HealthyHostCount
RequestCount
UnHealthyHostCount

Application ELB

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount
EstimatedALBConsumedLCUs
EstimatedALBNewConnectionCount
EstimatedProcessedBytes
HTTPCode_Backend_4XX
HTTPCode_Backend_5XX
HealthyHostCount
Latency
RequestCount
SurgeQueueLength
UnHealthyHostCount

Amazon EC2 Auto Scaling groups

CloudWatch Application Insights supports the following metrics:

CPUCreditBalance
CPUCreditUsage
CPUSurplusCreditBalance
CPUSurplusCreditsCharged
CPUUtilization
DiskReadBytes
DiskReadOps
DiskWriteBytes
DiskWriteOps
EBSByteBalance%
EBSIOBalance%
EBSReadBytes
EBSReadOps
EBSWriteBytes
EBSWriteOps
NetworkIn
NetworkOut
NetworkPacketsIn
NetworkPacketsOut
StatusCheckFailed
StatusCheckFailed_Instance
StatusCheckFailed_System

Amazon Simple Queue Server (SQS)

CloudWatch Application Insights supports the following metrics:

ApproximateAgeOfOldestMessage
ApproximateNumberOfMessagesDelayed
ApproximateNumberOfMessagesNotVisible
ApproximateNumberOfMessagesVisible
NumberOfEmptyReceives
NumberOfMessagesDeleted
NumberOfMessagesReceived

NumberOfMessagesSent

Amazon Relational Database Service (RDS)

CloudWatch Application Insights supports the following metrics:

Metrics

- [RDS Database instances \(p. 703\)](#)
- [RDS Database clusters \(p. 703\)](#)

RDS Database instances

BurstBalance

CPUCreditBalance

CPUUtilization

DatabaseConnections

DiskQueueDepth

FailedSQLServerAgentJobsCount

FreeStorageSpace

FreeableMemory

NetworkReceiveThroughput

NetworkTransmitThroughput

ReadIOPS

ReadLatency

ReadThroughput

WriteIOPS

WriteLatency

WriteThroughput

RDS Database clusters

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

BufferCacheHitRatio

CPUUtilization
CommitLatency
CommitThroughput
DDLLatency
DDLThroughput
DMLLatency
DMLThroughput
DatabaseConnections
Deadlocks
DeleteLatency
DeleteThroughput
EngineUptime
FreeLocalStorage
FreeableMemory
InsertLatency
InsertThroughput
LoginFailures
NetworkReceiveThroughput
NetworkThroughput
NetworkTransmitThroughput
Queries
ResultSetCacheHitRatio
SelectLatency
SelectThroughput
SnapshotStorageUsed
TotalBackupStorageBilled
UpdateLatency
UpdateThroughput
VolumeBytesUsed
VolumeReadIOPs
VolumeWriteIOPs

AWS Lambda function

CloudWatch Application Insights supports the following metrics:

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

Amazon DynamoDB table

CloudWatch Application Insights supports the following metrics:

SystemErrors

UserErrors

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

Amazon S3 bucket

CloudWatch Application Insights supports the following metrics:

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests
PutRequests
DeleteRequests
HeadRequests
PostRequests
SelectRequests
ListRequests
SelectScannedBytes
SelectReturnedBytes
FirstByteLatency
TotalRequestLatency
BytesDownloaded
BytesUploaded

AWS Step Functions

CloudWatch Application Insights supports the following metrics:

Metrics

- [Execution-level \(p. 706\)](#)
- [Activity \(p. 706\)](#)
- [Lambda function \(p. 707\)](#)
- [Service integration \(p. 707\)](#)
- [Step Functions API \(p. 707\)](#)

Execution-level

ExecutionTime
ExecutionThrottled
ExecutionsFailed
ExecutionsTimedOut
ExecutionsAborted
ExecutionsSucceeded
ExecutionsStarted

Activity

ActivityRunTime

ActivityScheduleTime
ActivityTime
ActivitiesFailed
ActivitiesHeartbeatTimedOut
ActivitiesTimedOut
ActivitiesScheduled
ActivitiesSucceeded
ActivitiesStarted

Lambda function

LambdaFunctionRunTime
LambdaFunctionScheduleTime
LambdaFunctionTime
LambdaFunctionsFailed
LambdaFunctionsTimedOut
LambdaFunctionsScheduled
LambdaFunctionsSucceeded
LambdaFunctionsStarted

Service integration

ServiceIntegrationRunTime
ServiceIntegrationScheduleTime
ServiceIntegrationTime
ServiceIntegrationsFailed
ServiceIntegrationsTimedOut
ServiceIntegrationsScheduled
ServiceIntegrationsSucceeded
ServiceIntegrationsStarted

Step Functions API

ThrottledEvents
ProvisionedBucketSize
ProvisionedRefillRate

ConsumedCapacity

API Gateway REST API stages

CloudWatch Application Insights supports the following metrics:

4XXError

5XXError

IntegrationLatency

Latency

DataProcessed

CacheHitCount

CacheMissCount

SAP HANA

CloudWatch Application Insights supports the following metrics:

hanadb_every_service_started_status

hanadb_daemon_service_started_status

hanadb_preprocessor_service_started_status

hanadb_webdispatcher_service_started_status

hanadb_compileserver_service_started_status

hanadb_nameserver_service_started_status

hanadb_server_startup_time_variations_seconds

hanadb_level_5_alerts_count

hanadb_level_4_alerts_count

hanadb_out_of_memory_events_count

hanadb_max_trigger_read_ratio_percent

hanadb_max_trigger_write_ratio_percent

hanadb_log_switch_wait_ratio_percent

hanadb_log_switch_race_ratio_percent

hanadb_time_since_last_savepoint_seconds

hanadb_disk_usage_highlevel_percent

hanadb_max_converter_page_number_count

hanadb_long_running_savepoints_count

hanadb_failed_io_reads_count
hanadb_failed_io_writes_count
hanadb_disk_data_unused_percent
hanadb_current_allocation_limit_used_percent
hanadb_table_allocation_limit_used_percent
hanadb_host_total_physical_memory_mb
hanadb_host_physical_memory_used_mb
hanadb_host_physical_memory_free_mb
hanadb_swap_memory_free_mb
hanadb_swap_memory_used_mb
hanadb_host_allocation_limit_mb
hanadb_host_total_memory_used_mb
hanadb_host_total_peak_memory_used_mb
hanadb_host_total_allocation_limit_mb
hanadb_host_code_size_mb
hanadb_host_shared_memory_allocation_mb
hanadb_cpu_usage_percent
hanadb_cpu_user_percent
hanadb_cpu_system_percent
hanadb_cpu_waitio_percent
hanadb_cpu_busy_percent
hanadb_cpu_idle_percent
hanadb_long_delta_merge_count
hanadb_unsuccessful_delta_merge_count
hanadb_successful_delta_merge_count
hanadb_row_store_allocated_size_mb
hanadb_row_store_free_size_mb
hanadb_row_store_used_size_mb
hanadb_temporary_tables_count
hanadb_large_non_compressed_tables_count
hanadb_total_non_compressed_tables_count

hanadb_longest_running_job_seconds
hanadb_average_commit_time_milliseconds
hanadb_suspended_sql_statements_count
hanadb_plan_cache_hit_ratio_percent
hanadb_plan_cache_lookup_count
hanadb_plan_cache_hit_count
hanadb_plan_cache_total_execution_microseconds
hanadb_plan_cache_cursor_duration_microseconds
hanadb_plan_cache_preparation_microseconds
hanadb_plan_cache_evicted_count
hanadb_plan_cache_evicted_microseconds
hanadb_plan_cache_evicted_preparation_count
hanadb_plan_cache_evicted_execution_count
hanadb_plan_cache_evicted_preparation_microseconds
hanadb_plan_cache_evicted_cursor_duration_microseconds
hanadb_plan_cache_evicted_total_execution_microseconds
hanadb_plan_cache_evicted_plan_size_mb
hanadb_plan_cache_count
hanadb_plan_cache_preparation_count
hanadb_plan_cache_execution_count
hanadb_network_collision_rate
hanadb_network_receive_rate
hanadb_network_transmit_rate
hanadb_network_packet_receive_rate
hanadb_network_packet_transmit_rate
hanadb_network_transmit_error_rate
hanadb_network_receive_error_rate
hanadb_time_until_license_expires_days
hanadb_is_license_valid_status
hanadb_local_running_connections_count
hanadb_local_idle_connections_count

hanadb_remote_running_connections_count
hanadb_remote_idle_connections_count
hanadb_last_full_data_backup_age_days
hanadb_last_data_backup_age_days
hanadb_last_log_backup_age_hours
hanadb_failed_data_backup_past_7_days_count
hanadb_failed_log_backup_past_7_days_count
hanadb_oldest_backup_in_catalog_age_days
hanadb_backup_catalog_size_mb
hanadb_hsr_replication_status
hanadb_hsr_log_shipping_delay_seconds
hanadb_hsr_secondary_failover_count
hanadb_hsr_secondary_reconnect_count
hanadb_hsr_async_buffer_used_mb
hanadb_hsr_secondary_active_status
hanadb_handle_count
hanadb_ping_time_milliseconds
hanadb_connection_count
hanadb_internal_connection_count
hanadb_external_connection_count
hanadb_idle_connection_count
hanadb_transaction_count
hanadb_internal_transaction_count
hanadb_external_transaction_count
hanadb_user_transaction_count
hanadb_blocked_transaction_count
hanadb_statement_count
hanadb_active_commit_id_range_count
hanadb_mvcc_version_count
hanadb_pending_session_count
hanadb_record_lock_count

hanadb_read_count
hanadb_write_count
hanadb_merge_count
hanadb_unload_count
hanadb_active_thread_count
hanadb_waiting_thread_count
hanadb_total_thread_count
hanadb_active_sql_executor_count
hanadb_waiting_sql_executor_count
hanadb_total_sql_executor_count
hanadb_data_write_size_mb
hanadb_data_write_time_milliseconds
hanadb_log_write_size_mb
hanadb_log_write_time_milliseconds
hanadb_data_read_size_mb
hanadb_data_read_time_milliseconds
hanadb_log_read_size_mb
hanadb_log_read_time_milliseconds
hanadb_data_backup_write_size_mb
hanadb_data_backup_write_time_milliseconds
hanadb_log_backup_write_size_mb
hanadb_log_backup_write_time_milliseconds
hanadb_mutex_collision_count
hanadb_read_write_lock_collision_count
hanadb_admission_control_admit_count
hanadb_admission_control_reject_count
hanadb_admission_control_queue_size_mb
hanadb_admission_control_wait_time_milliseconds

HA Cluster

CloudWatch Application Insights supports the following metrics:

ha_cluster_pacemaker_stonith_enabled

ha_cluster_corosync_quorate
hanadb_webdispatcher_service_started_status
ha_cluster_pacemaker_nodes
ha_cluster_corosync_ring_errors
ha_cluster_pacemaker_fail_count

Java

CloudWatch Application Insights supports the following metrics:

java_lang_memory_heapmemoryusage_used
java_lang_memory_heapmemoryusage_committed
java_lang_operatingsystem_openfiledescriptorcount
java_lang_operatingsystem_maxfiledescriptorcount
java_lang_operatingsystem_freephysicalmemorysize
java_lang_operatingsystem_freeswapspace size
java_lang_threading_threadcount
java_lang_threading_daemonthreadcount
java_lang_classloading_loadedclasscount
java_lang_garbagecollector_collectiontime_copy
java_lang_garbagecollector_collectiontime_ps_scavenge
java_lang_garbagecollector_collectiontime_parnew
java_lang_garbagecollector_collectiontime_marksweepcompact
java_lang_garbagecollector_collectiontime_ps_marksweep
java_lang_garbagecollector_collectiontime_concurrentmarksweep
java_lang_garbagecollector_collectiontime_g1_young_generation
java_lang_garbagecollector_collectiontime_g1_old_generation
java_lang_garbagecollector_collectiontime_g1_mixed_generation
java_lang_operatingsystem_committedvirtualmemorysize

Amazon Elastic Container Service (Amazon ECS)

CloudWatch Application Insights supports the following metrics:

Metrics

- [CloudWatch built-in metrics \(p. 714\)](#)
- [Container Insights metrics \(p. 714\)](#)

- [Container Insights Prometheus metrics \(p. 715\)](#)

CloudWatch built-in metrics

CPUReservation

CPUUtilization

MemoryReservation

MemoryUtilization

GPUReservation

Container Insights metrics

ContainerInstanceCount

CpuUtilized

CpuReserved

DeploymentCount

DesiredTaskCount

MemoryUtilized

MemoryReserved

NetworkRxBytes

NetworkTxBytes

PendingTaskCount

RunningTaskCount

ServiceCount

StorageReadBytes

StorageWriteBytes

TaskCount

TaskSetCount

instance_cpu_limit

instance_cpu_reserved_capacity

instance_cpu_usage_total

instance_cpu_utilization

instance_filesystem_utilization

instance_memory_limit

instance_memory_reserved_capacity
instance_memory_utilization
instance_memory_working_set
instance_network_total_bytes
instance_number_of_running_tasks

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used
java_lang_memory_heapmemoryusage_committed
java_lang_operatingsystem_openfiledescriptorcount
java_lang_operatingsystem_maxfiledescriptorcount
java_lang_operatingsystem_freephysicalmemorysize
java_lang_operatingsystem_freeswapspace size
java_lang_threading_threadcount
java_lang_classloading_loadedclasscount
java_lang_threading_daemonthreadcount
java_lang_garbagecollector_collectiontime_copy
java_lang_garbagecollector_collectiontime_ps_scavenge
java_lang_garbagecollector_collectiontime_parnew
java_lang_garbagecollector_collectiontime_marksweepcompact
java_lang_garbagecollector_collectiontime_ps_marksweep
java_lang_garbagecollector_collectiontime_concurrentmarksweep
java_lang_garbagecollector_collectiontime_g1_young_generation
java_lang_garbagecollector_collectiontime_g1_old_generation
java_lang_garbagecollector_collectiontime_g1_mixed_generation
java_lang_operatingsystem_committedvirtualmemorysize

Kubernetes on AWS

CloudWatch Application Insights supports the following metrics:

Metrics

- [Container Insights metrics \(p. 716\)](#)
- [Container Insights Prometheus metrics \(p. 716\)](#)

Container Insights metrics

cluster_failed_node_count
cluster_node_count
namespace_number_of_running_pods
node_cpu_limit
node_cpu_reserved_capacity
node_cpu_usage_total
node_cpu_utilization
node_filesystem_utilization
node_memory_limit
node_memory_reserved_capacity
node_memory_utilization
node_memory_working_set
node_network_total_bytes
node_number_of_running_containers
node_number_of_running_pods
pod_cpu_reserved_capacity
pod_cpu_utilization
pod_cpu_utilization_over_pod_limit
pod_memory_reserved_capacity
pod_memory_utilization
pod_memory_utilization_over_pod_limit
pod_network_rx_bytes
pod_network_tx_bytes
service_number_of_running_pods

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used
java_lang_memory_heapmemoryusage_committed
java_lang_operatingsystem_openfiledescriptorcount
java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize
java_lang_operatingsystem_freeswapspace size
java_lang_threading_threadcount
java_lang_classloading_loadedclasscount
java_lang_threading_daemonthreadcount
java_lang_garbagecollector_collectiontime_copy
java_lang_garbagecollector_collectiontime_ps_scavenge
java_lang_garbagecollector_collectiontime_parnew
java_lang_garbagecollector_collectiontime_marksweepcompact
java_lang_garbagecollector_collectiontime_ps_marksweep
java_lang_garbagecollector_collectiontime_concurrentmarksweep
java_lang_garbagecollector_collectiontime_g1_young_generation
java_lang_garbagecollector_collectiontime_g1_old_generation
java_lang_garbagecollector_collectiontime_g1_mixed_generation
java_lang_operatingsystem_committedvirtualmemorysize

Amazon FSx

CloudWatch Application Insights supports the following metrics:

DataReadBytes
DataWriteBytes
DataReadOperations
DataWriteOperations
MetadataOperations
FreeStorageCapacity
FreeDataStorageCapacity
LogicalDiskUsage
PhysicalDiskUsage

Metrics with data points requirements

For metrics without an obvious default threshold to alarm on, Application Insights waits until the metric has enough data points to predict a reasonable threshold to alarm on. The metric data points requirement that CloudWatch Application Insights checks before an alarm is created are:

- The metric has at least 100 data points from the past 15 to the past 2 days.
- The metric has at least 100 data points from the last day.

The following metrics follow these data points requirements. Note that CloudWatch agent metrics require up to one hour to create alarms.

Metrics

- [AWS/ApplicationELB \(p. 718\)](#)
- [AWS/AutoScaling \(p. 718\)](#)
- [AWS/EC2 \(p. 719\)](#)
- [Elastic Block Store \(EBS\) \(p. 719\)](#)
- [AWS/ELB \(p. 720\)](#)
- [AWS/RDS \(p. 720\)](#)
- [AWS/Lambda \(p. 721\)](#)
- [AWS/SQS \(p. 722\)](#)
- [AWS/CWAgent \(p. 722\)](#)
- [AWS/DynamoDB \(p. 723\)](#)
- [AWS/S3 \(p. 724\)](#)
- [AWS/States \(p. 724\)](#)
- [AWS/ApiGateway \(p. 725\)](#)
- [AWS/SNS \(p. 725\)](#)

AWS/ApplicationELB

ActiveConnectionCount

ConsumedLCUs

HTTPCode_ELB_4XX_Count

HTTPCode_Target_2XX_Count

HTTPCode_Target_3XX_Count

HTTPCode_Target_4XX_Count

HTTPCode_Target_5XX_Count

NewConnectionCount

ProcessedBytes

TargetResponseTime

UnHealthyHostCount

AWS/AutoScaling

GroupDesiredCapacity

GroupInServiceInstances

GroupMaxSize

GroupMinSize

GroupPendingInstances

GroupStandbyInstances

GroupTerminatingInstances

GroupTotalInstances

AWS/EC2

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

Elastic Block Store (EBS)

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumIdleTime

VolumeQueueLength
VolumeThroughputPercentage
VolumeConsumedReadWriteOps
BurstBalance

AWS/ELB

EstimatedALBActiveConnectionCount
EstimatedALBConsumedLCUs
EstimatedALBNewConnectionCount
EstimatedProcessedBytes
HTTPCode_Backend_4XX
HTTPCode_Backend_5XX
HealthyHostCount
Latency
RequestCount
SurgeQueueLength
UnHealthyHostCount

AWS/RDS

ActiveTransactions
AuroraBinlogReplicaLag
AuroraReplicaLag
BackupRetentionPeriodStorageUsed
BinLogDiskUsage
BlockedTransactions
CPUCreditBalance
CommitLatency
CommitThroughput
DDLLatency
DDLThroughput
DMLLatency
DMLThroughput
DatabaseConnections

Deadlocks
DeleteLatency
DeleteThroughput
DiskQueueDepth
EngineUptime
FreeLocalStorage
FreeStorageSpace
FreeableMemory
InsertLatency
InsertThroughput
LoginFailures
NetworkReceiveThroughput
NetworkThroughput
NetworkTransmitThroughput
Queries
ReadIOPS
ReadThroughput
SelectLatency
SelectThroughput
SnapshotStorageUsed
TotalBackupStorageBilled
UpdateLatency
UpdateThroughput
VolumeBytesUsed
VolumeReadIOPs
VolumeWriteIOPs
WriteIOPS
WriteThroughput

AWS/Lambda

Errors
DeadLetterErrors

Duration
Throttles
IteratorAge
ProvisionedConcurrencySpilloverInvocations

AWS/SQS

ApproximateAgeOfOldestMessage
ApproximateNumberOfMessagesDelayed
ApproximateNumberOfMessagesNotVisible
ApproximateNumberOfMessagesVisible
NumberOfEmptyReceives
NumberOfMessagesDeleted
NumberOfMessagesReceived
NumberOfMessagesSent

AWS/CWAgent

LogicalDisk % Free Space
Memory % Committed Bytes In Use
Memory Available Mbytes
Network Interface Bytes Total/sec
Paging File % Usage
PhysicalDisk % Disk Time
PhysicalDisk Avg. Disk sec/Read
PhysicalDisk Avg. Disk sec/Write
PhysicalDisk Disk Read Bytes/sec
PhysicalDisk Disk Reads/sec
PhysicalDisk Disk Write Bytes/sec
PhysicalDisk Disk Writes/sec
Processor % Idle Time
Processor % Interrupt Time
Processor % Processor Time
Processor % User Time
SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Page Splits/sec
SQLServer:Buffer Manager Buffer cache hit ratio
SQLServer:Buffer Manager Page life expectancy
SQLServer:Database Replica File Bytes Received/sec
SQLServer:Database Replica Log Bytes Received/sec
SQLServer:Database Replica Log remaining for undo
SQLServer:Database Replica Log Send Queue
SQLServer:Database Replica Mirrored Write Transaction/sec
SQLServer:Database Replica Recovery Queue
SQLServer:Database Replica Redo Bytes Remaining
SQLServer:Database Replica Redone Bytes/sec
SQLServer:Database Replica Total Log requiring undo
SQLServer:Database Replica Transaction Delay
SQLServer:General Statistics Processes blocked
SQLServer:SQL Statistics Batch Requests/sec
SQLServer:SQL Statistics SQL Compilations/sec
SQLServer:SQL Statistics SQL Re-Compilations/sec
System Processor Queue Length
TCPv4 Connections Established
TCPv6 Connections Established

AWS/DynamoDB

ConsumedReadCapacityUnits
ConsumedWriteCapacityUnits
ReadThrottleEvents
WriteThrottleEvents
TimeToLiveDeletedItemCount
ConditionalCheckFailedRequests
TransactionConflict
ReturnedRecordsCount
PendingReplicationCount
ReplicationLatency

AWS/S3

ReplicationLatency
BytesPendingReplication
OperationsPendingReplication
4xxErrors
5xxErrors
AllRequests
GetRequests
PutRequests
DeleteRequests
HeadRequests
PostRequests
SelectRequests
ListRequests
SelectScannedBytes
SelectReturnedBytes
FirstByteLatency
TotalRequestLatency
BytesDownloaded
BytesUploaded

AWS/States

ActivitiesScheduled
ActivitiesStarted
ActivitiesSucceeded
ActivityScheduleTime
ActivityRuntime
ActivityTime
LambdaFunctionsScheduled
LambdaFunctionsStarted
LambdaFunctionsSucceeded
LambdaFunctionScheduleTime

LambdaFunctionRuntime
LambdaFunctionTime
ServiceIntegrationsScheduled
ServiceIntegrationsStarted
ServiceIntegrationsSucceeded
ServiceIntegrationScheduleTime
ServiceIntegrationRuntime
ServiceIntegrationTime
ProvisionedRefillRate
ProvisionedBucketSize
ConsumedCapacity
ThrottledEvents

AWS/ApiGateway

4XXError
IntegrationLatency
Latency
DataProcessed
CacheHitCount
CacheMissCount

AWS/SNS

NumberOfNotificationsDelivered
NumberOfMessagesPublished
NumberOfNotificationsFailed
NumberOfNotificationsFilteredOut
NumberOfNotificationsFilteredOut-InvalidAttributes
NumberOfNotificationsFilteredOut-NoMessageAttributes
NumberOfNotificationsRedrivenToDlq
NumberOfNotificationsFailedToRedriveToDlq
SMSSuccessRate

Recommended metrics

The following table lists the recommended metrics for each component type.

Component type	Workload type	Recommended metric
EC2 instance (Windows servers)	Java Application	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_commi java_lang_operatingsystem_freephysicalmemo java_lang_operatingsystem_freeswapspace size
	Microsoft IIS/.NET Web Front-End	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC ASP.NET Applications Requests in Application Queue ASP.NET Requests Queued ASP.NET Application Restarts
	Microsoft SQL Server Database Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes

Component type	Workload type	Recommended metric
		Paging File % Usage System Processor Queue Length Network Interface Bytes Total/Sec PhysicalDisk % Disk Time SQLServer:Buffer Manager Buffer Cache Hit ratio SQLServer:Buffer Manager Page Life Expectancy SQLServer:General Statistics Processes Blocked SQLServer:General Statistics User Connections SQLServer:Locks Number of Deadlocks/Sec SQLServer:SQL Statistics Batch Requests/Sec
	.NET workerpool/Mid-Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC
	.NET Core Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes

Component type	Workload type	Recommended metric
	Active Directory	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Database ==> Instances Database Cache % Hit DirectoryServices DRA Pending Replication Operations DirectoryServices DRA Pending Replication Synchronizations DNS Recursive Query Failure/sec LogicalDisk Avg. Disk Queue Length

Component type	Workload type	Recommended metric
	SharePoint	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes ASP.NET Applications Cache API trims ASP.NET Requests Rejected ASP.NET Worker Process Restarts Memory Pages/sec SharePoint Publishing Cache Publishing cache flushes / second SharePoint Foundation Executing Time/Page Request SharePoint Disk-Based Cache Total number of cache compactions SharePoint Disk-Based Cache Blob cache hit ratio SharePoint Disk-Based Cache Blob Cache fill ratio SharePoint Disk-Based Cache Blob cache flushes / second ASP.NET Requests Queued ASP.NET Applications Requests in Application Queue ASP.NET Application Restarts LogicalDisk Avg. Disk sec/Write LogicalDisk Avg. Disk sec/Read Processor % Interrupt Time

Component type	Workload type	Recommended metric
EC2 instance (Linux servers)	Java Application	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_commi java_lang_operatingsystem_freephysicalmemo java_lang_operatingsystem_freeswapspace size
	.NET Core Tier or SQL Server Database Tier	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent
EC2 instance group	SAP HANA multi-node or single node	<ul style="list-style-type: none"> • hanadb_server_startup_time_variations_seconds • hanadb_level_5_alerts_count • hanadb_level_4_alerts_count • hanadb_out_of_memory_events_count • hanadb_max_trigger_read_ratio_percent • hanadb_max_trigger_write_ratio_percent • hanadb_log_switch_race_ratio_percent • hanadb_time_since_last_savepoint_seconds • hanadb_disk_usage_highlevel_percent • hanadb_current_allocation_limit_used_percent • hanadb_table_allocation_limit_used_percent • hanadb_cpu_usage_percent • hanadb_plan_cache_hit_ratio_percent • hanadb_last_data_backup_age_days

Component type	Workload type	Recommended metric
EBS volume	Any	VolumeReadBytes VolumeWriteBytes VolumeReadOps VolumeWriteOps VolumeQueueLength VolumeThroughputPercentage VolumeConsumedReadWriteOps BurstBalance
Classic ELB	Any	HTTPCode_Backend_4XX HTTPCode_Backend_5XX Latency SurgeQueueLength UnHealthyHostCount
Application ELB	Any	HTTPCode_Target_4XX_Count HTTPCode_Target_5XX_Count TargetResponseTime UnHealthyHostCount
RDS Database instance	Any	CPUUtilization ReadLatency WriteLatency BurstBalance FailedSQLServerAgentJobsCount
RDS Database cluster	Any	CPUUtilization CommitLatency DatabaseConnections Deadlocks FreeableMemory NetworkThroughput VolumeBytesUsed

Component type	Workload type	Recommended metric
Lambda Function	Any	Duration Errors IteratorAge ProvisionedConcurrencySpilloverInvocations Throttles
SQS Queue	Any	ApproximateAgeOfOldestMessage ApproximateNumberOfMessagesVisible NumberOfMessagesSent
Amazon DynamoDB table	Any	SystemErrors UserErrors ConsumedReadCapacityUnits ConsumedWriteCapacityUnits ReadThrottleEvents WriteThrottleEvents ConditionalCheckFailedRequests TransactionConflict
Amazon S3 bucket	Any	If replication configuration with Replication Time Control (RTC) is enabled: ReplicationLatency BytesPendingReplication OperationsPendingReplication If request metrics are turned on: 5xxErrors 4xxErrors BytesDownloaded BytesUploaded

Component type	Workload type	Recommended metric
AWS Step Functions	Any	<p>General</p> <ul style="list-style-type: none"> • ExecutionThrottled • ExecutionsAborted • ProvisionedBucketSize • ProvisionedRefillRate • ConsumedCapacity <p>If state machine type is EXPRESS or log group level is OFF</p> <ul style="list-style-type: none"> • ExecutionsFailed • ExecutionsTimedOut <p>If state machine has Lambda functions</p> <ul style="list-style-type: none"> • LambdaFunctionsFailed • LambdaFunctionsTimedOut <p>If state machine has activities</p> <ul style="list-style-type: none"> • ActivitiesFailed • ActivitiesTimedOut • ActivitiesHeartbeatTimedOut <p>If state machine has service integrations</p> <ul style="list-style-type: none"> • ServiceIntegrationsFailed • ServiceIntegrationsTimedOut
API Gateway REST API stage	Any	<ul style="list-style-type: none"> • 4XXErrors • 5XXErrors • Latency

Component type	Workload type	Recommended metric
ECS Cluster	Any	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 Launch Type only) CPUUtilization (EC2 Launch Type only) MemoryReservation (EC2 Launch Type only) MemoryUtilization (EC2 Launch Type only) GPUReservation (EC2 Launch Type only) instance_cpu_utilization (EC2 Launch Type only) instance_filesystem_utilization (EC2 Launch Type only) instance_memory_utilization (EC2 Launch Type only) instance_network_total_bytes (EC2 Launch Type only)

Component type	Workload type	Recommended metric
	Java Application	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 Launch Type only) CPUUtilization (EC2 Launch Type only) MemoryReservation (EC2 Launch Type only) MemoryUtilization (EC2 Launch Type only) GPUReservation (EC2 Launch Type only) instance_cpu_utilization (EC2 Launch Type only) instance_filesystem_utilization (EC2 Launch Type only) instance_memory_utilization (EC2 Launch Type only) instance_network_total_bytes (EC2 Launch Type only) java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_commi java_lang_operatingsystem_freephysicalmem java_lang_operatingsystem_freeswapspace size

Component type	Workload type	Recommended metric
ECS Service	Any	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes
	Java Application	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_commi java_lang_operatingsystem_freephysicalmem java_lang_operatingsystem_freeswapspace

Component type	Workload type	Recommended metric
EKS Cluster	Any	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_commi java_lang_operatingsystem_freephysicalmem java_lang_operatingsystem_freeswapspace size

Component type	Workload type	Recommended metric
Kubernetes Cluster on EC2	Any	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_commi java_lang_operatingsystem_freephysicalmem java_lang_operatingsystem_freeswapspace size

Performance Counter metrics

Performance Counter metrics are recommended for instances only when the corresponding Performance Counter sets are installed on the Windows instances.

Performance Counter metric name	Performance Counter set name
.NET CLR Exceptions # of Exceps Thrown	.NET CLR Exceptions
.NET CLR Exceptions # of Exceps Thrown/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Filters/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Finallys/Sec	.NET CLR Exceptions

Performance Counter metric name	Performance Counter set name
.NET CLR Exceptions Throw to Catch Depth/Sec	.NET CLR Exceptions
.NET CLR Interop # of CCWs	.NET CLR Interop
.NET CLR Interop # of Stubs	.NET CLR Interop
.NET CLR Interop # of TLB exports/Sec	.NET CLR Interop
.NET CLR Interop # of TLB imports/Sec	.NET CLR Interop
.NET CLR Interop # of Marshaling	.NET CLR Interop
.NET CLR Jit % Time in Jit	.NET CLR Jit
.NET CLR Jit Standard Jit Failures	.NET CLR Jit
.NET CLR Loading % Time Loading	.NET CLR Loading
.NET CLR Loading Rate of Load Failures	.NET CLR Loading
.NET CLR LocksAndThreads Contention Rate/Sec	.NET CLR LocksAndThreads
.NET CLR LocksAndThreads Queue Length/Sec	.NET CLR LocksAndThreads
.NET CLR Memory # Total Committed Bytes	.NET CLR Memory
.NET CLR Memory % Time in GC	.NET CLR Memory
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/Sec	.NET CLR Networking 4.0.0.0
APP_POOL_WAS Total Worker Process Ping Failures	APP_POOL_WAS
ASP.NET Application Restarts	ASP.NET
ASP.NET Requests Rejected	ASP.NET
ASP.NET Worker Process Restarts	ASP.NET
ASP.NET Applications Cache API trims	ASP.NET Applications
ASP.NET Applications % Managed Processor Time (estimated)	ASP.NET Applications
ASP.NET Applications Errors Total/Sec	ASP.NET Applications
ASP.NET Applications Errors Unhandled During Execution/Sec	ASP.NET Applications

Performance Counter metric name	Performance Counter set name
ASP.NET Applications Requests in Application Queue	ASP.NET Applications
ASP.NET Applications Requests/Sec	ASP.NET Applications
ASP.NET Request Wait Time	ASP.NET
ASP.NET Requests Queued	ASP.NET
Database ==> Instances Database Cache % Hit	Database ==> Instances
Database ==> Instances I/O Database Reads Average Latency	Database ==> Instances
Database ==> Instances I/O Database Reads/sec	Database ==> Instances
Database ==> Instances I/O Log Writes Average Latency	Database ==> Instances
DirectoryServices DRA Pending Replication Operations	DirectoryServices
DirectoryServices DRA Pending Replication Synchronizations	DirectoryServices
DirectoryServices LDAP Bind Time	DirectoryServices
DNS Recursive Queries/sec	DNS
DNS Recursive Query Failure/sec	DNS
DNS TCP Query Received/sec	DNS
DNS Total Query Received/sec	DNS
DNS Total Response Sent/sec	DNS
DNS UDP Query Received/sec	DNS
HTTP Service Request Queues CurrentQueueSize	HTTP Service Request Queues
LogicalDisk % Free Space	LogicalDisk
LogicalDisk Avg. Disk sec/Write	LogicalDisk
LogicalDisk Avg. Disk sec/Read	LogicalDisk
LogicalDisk Avg. Disk Queue Length	LogicalDisk
Memory % Committed Bytes In Use	Memory
Memory Available Mbytes	Memory
Memory Pages/Sec	Memory
Memory Long-Term Average Standby Cache Lifetime (s)	Memory
Network Interface Bytes Total/Sec	Network Interface
Network Interface Bytes Received/sec	Network Interface

Performance Counter metric name	Performance Counter set name
Network Interface Bytes Sent/sec	Network Interface
Network Interface Current Bandwidth	Network Interface
Paging File % Usage	Paging File
PhysicalDisk % Disk Time	PhysicalDisk
PhysicalDisk Avg. Disk Queue Length	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Read	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Write	PhysicalDisk
PhysicalDisk Disk Read Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Reads/Sec	PhysicalDisk
PhysicalDisk Disk Write Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Writes/Sec	PhysicalDisk
Processor % Idle Time	Processor
Processor % Interrupt Time	Processor
Processor % Processor Time	Processor
Processor % User Time	Processor
SharePoint Disk-Based Cache Blob Cache fill ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache flushes / second	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache hit ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Total number of cache compactions	SharePoint Disk-Based Cache
SharePoint Foundation Executing Time/Page Request	SharePoint Foundation
SharePoint Publishing Cache Publishing cache flushes / second	SharePoint Publishing Cache
Security System-Wide Statistics Kerberos Authentications	Security System-Wide Statistics
Security System-Wide Statistics NTLM Authentications	Security System-Wide Statistics
SQLServer:Access Methods Forwarded Records/ Sec	SQLServer:Access Methods
SQLServer:Access Methods Full Scans/Sec	SQLServer:Access Methods
SQLServer:Access Methods Page Splits/Sec	SQLServer:Access Methods
SQLServer:Buffer Manager Buffer cache hit Ratio	SQLServer:Buffer Manager

Performance Counter metric name	Performance Counter set name
SQLServer:Buffer Manager Page life Expectancy	SQLServer:Buffer Manager
SQLServer:Database Replica File Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log remaining for undo	SQLServer:Database Replica
SQLServer:Database Replica Log Send Queue	SQLServer:Database Replica
SQLServer:Database Replica Mirrored Write Transaction/sec	SQLServer:Database Replica
SQLServer:Database Replica Recovery Queue	SQLServer:Database Replica
SQLServer:Database Replica Redo Bytes Remaining	SQLServer:Database Replica
SQLServer:Database Replica Redone Bytes/sec	SQLServer:Database Replica
SQLServer:Database Replica Total Log requiring undo	SQLServer:Database Replica
SQLServer:Database Replica Transaction Delay	SQLServer:Database Replica
SQLServer:General Statistics Processes Blocked	SQLServer:General Statistics
SQLServer:General Statistics User Connections	SQLServer:General Statistics
SQLServer:Latches Average Latch Wait Time (ms)	SQLServer:Latches
SQLServer:Locks Average Wait Time (ms)	SQLServer:Locks
SQLServer:Locks Lock Timeouts/Sec	SQLServer:Locks
SQLServer:Locks Lock Waits/Sec	SQLServer:Locks
SQLServer:Locks Number of Deadlocks/Sec	SQLServer:Locks
SQLServer:Memory Manager Memory Grants Pending	SQLServer:Memory Manager
SQLServer:SQL Statistics Batch Requests/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Compilations/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Re-Compilations/Sec	SQLServer:SQL Statistics
System Processor Queue Length	System
TCPv4 Connections Established	TCPv4
TCPv6 Connections Established	TCPv6
W3SVC_W3WP File Cache Flushes	W3SVC_W3WP

Performance Counter metric name	Performance Counter set name
W3SVC_W3WP File Cache Misses	W3SVC_W3WP
W3SVC_W3WP Requests/Sec	W3SVC_W3WP
W3SVC_W3WP URI Cache Flushes	W3SVC_W3WP
W3SVC_W3WP URI Cache Misses	W3SVC_W3WP
Web Service Bytes Received/Sec	Web Service
Web Service Bytes Sent/Sec	Web Service
Web Service Connection Attempts/Sec	Web Service
Web Service Current Connections	Web Service
Web Service Get Requests/Sec	Web Service
Web Service Post Requests/Sec	Web Service

AWS services that publish CloudWatch metrics

The following AWS services publish metrics to CloudWatch. For information about the metrics and dimensions, see the specified documentation.

Service	Namespace	Documentation
AWS Amplify	AWS/ AmplifyHosting	Monitoring
Amazon API Gateway	AWS/ApiGateway	Monitor API Execution with Amazon CloudWatch
AWS App Runner	AWS/AppRunner	Viewing App Runner service metrics reported to CloudWatch
AppStream 2.0	AWS/AppStream	Monitoring Amazon AppStream 2.0 Resources
AWS AppSync	AWS/AppSync	CloudWatch Metrics
Amazon Athena	AWS/Athena	Monitoring Athena Queries with CloudWatch Metrics
Amazon Aurora	AWS/RDS	Amazon Aurora metrics
AWS Backup	AWS/Backup	Monitoring AWS Backup Metrics with CloudWatch
AWS Billing and Cost Management	AWS/Billing	Monitoring Charges with Alerts and Notifications
AWS Certificate Manager	AWS/ CertificateManager	Supported CloudWatch Metrics
ACM Private CA	AWS/ ACMPrivateCA	Supported CloudWatch Metrics
AWS Chatbot	AWS/Chatbot	Monitoring AWS Chatbot with Amazon CloudWatch
AWS Client VPN	AWS/ClientVPN	Monitoring with Amazon CloudWatch
Amazon CloudFront	AWS/CloudFront	Monitoring CloudFront Activity Using CloudWatch
AWS CloudHSM	AWS/CloudHSM	Getting CloudWatch Metrics
Amazon CloudSearch	AWS/ CloudSearch	Monitoring an Amazon CloudSearch Domain with Amazon CloudWatch
CloudWatch agent	CWAgent or a custom namespace	Metrics collected by the CloudWatch agent
CloudWatch metric streams	AWS/ CloudWatch/ MetricStreams	Monitoring your metric streams with CloudWatch metrics

Service	Namespace	Documentation
CloudWatch RUM	AWS/RUM	CloudWatch metrics that you can collect with CloudWatch RUM
CloudWatch Synthetics	CloudWatchSynthetics	CloudWatch metrics published by canaries
Amazon CloudWatch Logs	AWS/Logs	Monitoring Usage with CloudWatch Metrics
AWS CodeBuild	AWS/CodeBuild	Monitoring AWS CodeBuild
Amazon CodeGuru Profiler	AWS/CodeGuruProfiler	Monitoring Amazon CodeGuru Profiler with Amazon CloudWatch
Amazon Cognito	AWS/Cognito	Monitoring Amazon Cognito
Amazon Connect	AWS/Connect	Monitoring Amazon Connect in Amazon CloudWatch Metrics
Amazon Data Lifecycle Manager	AWS/DataLifecycleManager	Monitor your policies using Amazon CloudWatch
AWS DataSync	AWS/DataSync	Monitoring Your Task
AWS Database Migration Service	AWS/DMS	Monitoring AWS DMS Tasks
AWS Direct Connect	AWS/DX	Monitoring with Amazon CloudWatch
Amazon DocumentDB	AWS/DocDB	Amazon DocumentDB Metrics
Amazon DynamoDB	AWS/DynamoDB	DynamoDB Metrics and Dimensions
DynamoDB Accelerator (DAX)	AWS/DAX	Viewing DAX Metrics and Dimensions
Amazon EC2	AWS/EC2	Monitoring Your Instances Using CloudWatch
Amazon EC2 Elastic Graphics	AWS/ElasticGPUs	Using CloudWatch metrics to monitor Elastic Graphics
Amazon EC2 Spot Fleet	AWS/EC2Spot	CloudWatch Metrics for Spot Fleet
Amazon EC2 Auto Scaling	AWS/AutoScaling	Monitoring Your Auto Scaling Groups and Instances Using CloudWatch
AWS Elastic Beanstalk	AWS/ElasticBeanstalk	Publishing Amazon CloudWatch Custom Metrics for an Environment
Amazon Elastic Block Store	AWS/EBS	Amazon CloudWatch Metrics for Amazon EBS
Amazon Elastic Container Service	AWS/ECS	Amazon ECS CloudWatch Metrics

Service	Namespace	Documentation
Amazon ECS Cluster auto scaling	AWS/ECS/ManagedScaling	Amazon ECS cluster auto scaling
Amazon Elastic File System	AWS/EFS	Monitoring with CloudWatch
Amazon Elastic Inference	AWS/ElasticInference	Using CloudWatch Metrics to Monitor Amazon EI
Elastic Load Balancing	AWS/ApplicationELB	CloudWatch Metrics for your Application Load Balancer
Elastic Load Balancing	AWS/NetworkELB	CloudWatch Metrics for your Network Load Balancer
Elastic Load Balancing	AWS/GatewayELB	CloudWatch Metrics for your Gateway Load Balancer
Elastic Load Balancing	AWS/ELB	CloudWatch Metrics for your Classic Load Balancer
Amazon Elastic Transcoder	AWS/ElasticTranscoder	Monitoring with Amazon CloudWatch
Amazon ElastiCache for Memcached	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon ElastiCache for Redis	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon OpenSearch Service	AWS/ES	Monitoring Cluster Metrics with Amazon CloudWatch
Amazon EMR	AWS/ElasticMapReduce	Monitor Metrics with CloudWatch
AWS Elemental MediaConnect	AWS/MediaConnect	Monitoring MediaConnect with Amazon CloudWatch
AWS Elemental MediaConvert	AWS/MediaConvert	Using CloudWatch Metrics to View Metrics for AWS Elemental MediaConvert Resources
AWS Elemental MediaLive	AWS/MediaLive	Monitoring activity using Amazon CloudWatch metrics
AWS Elemental MediaPackage	AWS/MediaPackage	Monitoring AWS Elemental MediaPackage with Amazon CloudWatch Metrics
AWS Elemental MediaStore	AWS/MediaStore	Monitoring AWS Elemental MediaStore with Amazon CloudWatch Metrics
AWS Elemental MediaTailor	AWS/MediaTailor	Monitoring AWS Elemental MediaTailor with Amazon CloudWatch

Service	Namespace	Documentation
Amazon EventBridge	AWS/Events	Monitoring Amazon EventBridge
Amazon FSx for Lustre	AWS/FSx	Monitoring Amazon FSx for Lustre
Amazon FSx for Windows File Server	AWS/FSx	Monitoring Amazon FSx for Windows File Server
Amazon GameLift	AWS/GameLift	Monitor Amazon GameLift with CloudWatch
AWS Global Accelerator	AWS/GlobalAccelerator	Using Amazon CloudWatch with AWS Global Accelerator
AWS Glue	Glue	Monitoring AWS Glue Using CloudWatch Metrics
AWS Ground Station	AWS/GroundStation	Metrics Using Amazon CloudWatch
Amazon HealthLake	AWS/HealthLake	Monitoring HealthLake with CloudWatch
Amazon Inspector	AWS/Inspector	Monitoring Amazon Inspector Using CloudWatch
Amazon Interactive Video Service (IVS)	AWS/IVS	Monitoring Amazon IVS with Amazon CloudWatch
AWS IoT	AWS/IoT	AWS IoT Metrics and Dimensions
AWS IoT Analytics	AWS/IoTAnalytics	Namespace, Metrics, and Dimensions
AWS IoT SiteWise	AWS/IoTSiteWise	Monitoring AWS IoT SiteWise with Amazon CloudWatch metrics
AWS IoT Things Graph	AWS/ThingsGraph	Metrics
AWS Key Management Service	AWS/KMS	Monitoring with CloudWatch
Amazon Keyspaces (for Apache Cassandra)	AWS/Cassandra	Amazon Keyspaces Metrics and Dimensions
Amazon Kinesis Data Analytics	AWS/KinesisAnalytics	Kinesis Data Analytics for SQL Applications: Monitoring with CloudWatch Kinesis Data Analytics for Apache Flink: Viewing Amazon Kinesis Data Analytics Metrics and Dimensions
Amazon Kinesis Data Firehose	AWS/Firehose	Monitoring Kinesis Data Firehose Using CloudWatch Metrics
Amazon Kinesis Data Streams	AWS/Kinesis	Monitoring Amazon Kinesis Data Streams with Amazon CloudWatch

Service	Namespace	Documentation
Amazon Kinesis Video Streams	AWS/KinesisVideo	Monitoring Kinesis Video Streams Metrics with CloudWatch
AWS Lambda	AWS/Lambda	AWS Lambda Metrics
Amazon Lex	AWS/Lex	Monitoring Amazon Lex with Amazon CloudWatch
Amazon Location Service	AWS/Location	Amazon Location Service metrics exported to Amazon CloudWatch
Amazon Lookout for Metrics	AWS/LookoutMetrics	Monitoring Lookout for Metrics with Amazon CloudWatch
Amazon Machine Learning	AWS/ML	Monitoring Amazon ML with CloudWatch Metrics
Amazon Managed Streaming for Apache Kafka	AWS/Kafka	Monitoring Amazon MSK with Amazon CloudWatch
Amazon MQ	AWS/AmazonMQ	Monitoring Amazon MQ Brokers Using Amazon CloudWatch
Amazon Neptune	AWS/Neptune	Monitoring Neptune with CloudWatch
AWS Network Firewall	AWS/NetworkFirewall	AWS Network Firewall metrics in Amazon CloudWatch
AWS Network Manager	AWS/NetworkManager	CloudWatch metrics for on-premises resources
AWS OpsWorks	AWS/OpsWorks	Monitoring Stacks using Amazon CloudWatch
Amazon Polly	AWS/Polly	Integrating CloudWatch with Amazon Polly
AWS PrivateLink	AWS/PrivateLinkEndpoints	CloudWatch metrics for AWS PrivateLink
AWS PrivateLink	AWS/PrivateLinkServices	CloudWatch metrics for AWS PrivateLink
Amazon QLDB	AWS/QLDB	Amazon QLDB Metrics and Dimensions
Amazon Redshift	AWS/Redshift	Amazon Redshift Performance Data
Amazon Relational Database Service	AWS/RDS	Monitoring Amazon RDS metrics with Amazon CloudWatch
AWS RoboMaker	AWS/Robomaker	Monitoring AWS RoboMaker with Amazon CloudWatch
Amazon Route 53	AWS/Route53	Monitoring Amazon Route 53
Route 53 Application Recovery Controller	AWS/Route53RecoveryReadiness	Using Amazon CloudWatch with Application Recovery Readiness
Amazon SageMaker	AWS/SageMaker	Monitoring SageMaker with CloudWatch

Service	Namespace	Documentation
Amazon SageMaker Model Building Pipelines	AWS/SageMaker/ModelBuildingPipeline	SageMaker Pipelines Metrics
AWS SDK Metrics for Enterprise Support	AWS/SDKMetrics	Metrics and Data Collected by AWS SDK Metrics for Enterprise Support
AWS Service Catalog	AWS/ServiceCatalog	AWS Service Catalog CloudWatch Metrics
AWS Shield Advanced	AWS/DDoSProtection	Monitoring with CloudWatch
Amazon Simple Email Service	AWS/SES	Retrieving Amazon SES Event Data from CloudWatch
Amazon Simple Notification Service	AWS/SNS	Monitoring Amazon SNS with CloudWatch
Amazon Simple Queue Service	AWS/SQS	Monitoring Amazon SQS Queues Using CloudWatch
Amazon S3	AWS/S3	Monitoring Metrics with Amazon CloudWatch
S3 Storage Lens	AWS/S3/Storage-Lens	Monitor S3 Storage Lens metrics in CloudWatch
Amazon Simple Workflow Service	AWS/SWF	Amazon SWF Metrics for CloudWatch
AWS Step Functions	AWS/States	Monitoring Step Functions Using CloudWatch
AWS Storage Gateway	AWS/StorageGateway	Monitoring Your Gateway and Resources
AWS Systems Manager Run Command	AWS/SSM-RunCommand	Monitoring Run Command Metrics Using CloudWatch
Amazon Textract	AWS/Textract	CloudWatch Metrics for Amazon Textract
Amazon Timestream	AWS/Timestream	Timestream Metrics and Dimensions
AWS Transfer for SFTP	AWS/Transfer	AWS SFTP CloudWatch Metrics
Amazon Translate	AWS/Translate	CloudWatch Metrics and Dimensions for Amazon Translate
AWS Trusted Advisor	AWS/TrustedAdvisor	Creating Trusted Advisor Alarms Using CloudWatch
Amazon VPC	AWS/NATGateway	Monitoring Your NAT Gateway with CloudWatch
Amazon VPC	AWS/TransitGateway	CloudWatch Metrics for Your Transit Gateways

Service	Namespace	Documentation
Amazon VPC	AWS/VPN	Monitoring with CloudWatch
Amazon VPC IP Address Manager	AWS/IPAM	Create alarms with Amazon CloudWatch
AWS WAF	AWS/WAFV2 for AWS WAF resources WAF for AWS WAF classic resources	Monitoring with CloudWatch
Amazon WorkMail	AWS/WorkMail	Monitoring Amazon WorkMail with Amazon CloudWatch
Amazon WorkSpaces	AWS/WorkSpaces	Monitor Your WorkSpaces Using CloudWatch Metrics
Amazon WorkSpaces Web	AWS/ WorkSpacesWeb	Monitoring Amazon WorkSpaces Web with Amazon CloudWatch

Alarm events and EventBridge

CloudWatch sends events to Amazon EventBridge whenever a CloudWatch alarm is created, updated, deleted, or changes alarm state. You can use EventBridge and these events to write rules that take actions, such as notifying you, when an alarm changes state. For more information, see [What is Amazon EventBridge?](#)

CloudWatch guarantees the delivery of alarm state change events to EventBridge.

Sample events from CloudWatch

This section includes example events from CloudWatch.

State change for a single-metric alarm

```
{  
    "version": "0",  
    "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",  
    "detail-type": "CloudWatch Alarm State Change",  
    "source": "aws.cloudwatch",  
    "account": "123456789012",  
    "time": "2019-10-02T17:04:40Z",  
    "region": "us-east-1",  
    "resources": [  
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"  
    ],  
    "detail": {  
        "alarmName": "ServerCpuTooHigh",  
        "configuration": {  
            "description": "Goes into alarm when server CPU utilization is too high!",  
            "metrics": [  
                {  
                    "id": "30b6c6b2-a864-43a2-4877-c09a1afc3b87",  
                    "metricStat": {  
                        "metric": {  
                            "dimensions": {  
                                "InstanceId": "i-12345678901234567"  
                            },  
                            "name": "CPUUtilization",  
                            "namespace": "AWS/EC2"  
                        },  
                        "period": 300,  
                        "stat": "Average"  
                    },  
                    "returnData": true  
                }  
            ]  
        },  
        "previousState": {  
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints [0.0666851903306472 (01/10/19 13:46:00)] was not greater than the threshold (50.0) (minimum 1 datapoint for ALARM -> OK transition).",  
            "reasonData": "{\"version\":\"1.0\",\"queryDate\": \"2019-10-01T13:56:40.985+0000\", \"startDate\":\"2019-10-01T13:46:00.000+0000\", \"statistic\": \"Average\", \"period\":300, \"recentDatapoints\":[0.0666851903306472], \"threshold\":50.0}",  
            "timestamp": "2019-10-01T13:56:40.987+0000",  
            "value": "OK"  
        },  
    }  
}
```

```

        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints [99.501602296934344 (02/10/19 16:59:00)] was greater than the threshold (50.0) (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":\"2019-10-02T17:04:40.985+0000\",\"startDate\":\"2019-10-02T16:59:00.000+0000\",\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[99.50160229693434],\"threshold\":50.0}",
            "timestamp": "2019-10-02T17:04:40.989+0000",
            "value": "ALARM"
        }
    }
}

```

State change for a metric math alarm

```
{
    "version": "0",
    "id": "2dde0eb1-528b-d2d5-9ca6-6d590caf2329",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:20:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "configuration": {
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "metrics": [
                {
                    "expression": "SUM(METRICS())",
                    "id": "e1",
                    "label": "Total Network Traffic",
                    "returnData": true
                },
                {
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkIn",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkOut",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    }
                }
            ]
        }
    }
}
```

```

        },
        "returnData": false
    }
]
},
"previousState": {
    "reason": "Unchecked: Initial alarm creation",
    "timestamp": "2019-10-02T17:20:03.642+0000",
    "value": "INSUFFICIENT_DATA"
},
"state": {
    "reason": "Threshold Crossed: 1 out of the last 1 datapoints [45628.0 (02/10/19 17:10:00)] was greater than the threshold (10000.0) (minimum 1 datapoint for OK -> ALARM transition).",
    "reasonData": "{\"version\":\"1.0\",\"queryDate\":\\\"2019-10-02T17:20:48.551+0000\\\",\\\"startDate\\\":\\\"2019-10-02T17:10:00.000+0000\\\",\\\"period\\\":300,\\\"recentDatapoints\\\":[45628.0],\\\"threshold\\\":10000.0}",
    "timestamp": "2019-10-02T17:20:48.554+0000",
    "value": "ALARM"
}
}
}

```

State change for an anomaly detection alarm

```
{
    "version": "0",
    "id": "daafc9f1-bddd-c6c9-83af-74971fcfc4ef",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-03T16:00:04Z",
    "region": "us-east-1",
    "resources": ["arn:aws:cloudwatch:us-east-1:123456789012:alarm:EC2 CPU Utilization Anomaly"],
    "detail": {
        "alarmName": "EC2 CPU Utilization Anomaly",
        "state": {
            "value": "ALARM",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.0 (03/10/19 15:58:00)] was less than the lower thresholds [0.020599444741798756] or greater than the upper thresholds [0.3006915352732461] (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":\\\"2019-10-03T16:00:04.650+0000\\\",\\\"startDate\\\":\\\"2019-10-03T15:58:00.000+0000\\\",\\\"period\\\":60,\\\"recentDatapoints\\\":[0.0],\\\"recentLowerThresholds\\\": [0.020599444741798756],\\\"recentUpperThresholds\\\": [0.3006915352732461]}",
            "timestamp": "2019-10-03T16:00:04.653+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.166666666664241 (03/10/19 15:57:00)] was not less than the lower thresholds [0.0206719426210418] or not greater than the upper thresholds [0.30076870222143803] (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":\\\"2019-10-03T15:59:04.670+0000\\\",\\\"startDate\\\":\\\"2019-10-03T15:57:00.000+0000\\\",\\\"period\\\":60,\\\"recentDatapoints\\\": [0.166666666664241],\\\"recentLowerThresholds\\\": [0.0206719426210418],\\\"recentUpperThresholds\\\": [0.30076870222143803]}",
            "timestamp": "2019-10-03T15:59:04.672+0000"
        },
        "configuration": {
            "description": "Goes into alarm if CPU Utilization is out of band",
            "metrics": [
                {
                    "id": "m1",
                    "metricStat": {

```

```
        "metric": {
            "namespace": "AWS/EC2",
            "name": "CPUUtilization",
            "dimensions": {
                "InstanceId": "i-12345678901234567"
            },
            "period": 60,
            "stat": "Average"
        },
        "returnData": true
    },
    {
        "id": "ad1",
        "expression": "ANOMALY_DETECTION_BAND(m1, 0.8)",
        "label": "CPUUtilization (expected)",
        "returnData": true
    }
]
}
}
```

Creation of a metric alarm

```
{  
    "version": "0",  
    "id": "dc69d7c7-098a-506c-5b5e-4500ee7772ef",  
    "detail-type": "CloudWatch Alarm Configuration Change",  
    "source": "aws.cloudwatch",  
    "account": "123456789012",  
    "time": "2022-03-03T17:06:13Z",  
    "region": "us-east-1",  
    "resources": [  
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"  
    ],  
    "detail": {  
        "alarmName": "ServerCpuTooHigh",  
        "operation": "create",  
        "state": {  
            "value": "INSUFFICIENT_DATA",  
            "timestamp": "2022-03-03T17:06:13.757+0000"  
        },  
        "configuration": {  
            "evaluationPeriods": 1,  
            "threshold": 70,  
            "comparisonOperator": "GreaterThanThreshold",  
            "treatMissingData": "ignore",  
            "metrics": [  
                {  
                    "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",  
                    "metricStat": {  
                        "metric": {  
                            "namespace": "AWS/EC2",  
                            "name": "CPUUtilization",  
                            "dimensions": {  
                                "InstanceId": "i-12345678901234567"  
                            }  
                        },  
                        "period": 300,  
                        "stat": "Average"  
                    },  
                    "returnData": true  
                }  
            ],  
            "alarmName": "ServerCpuTooHigh",  
            "description": "Goes into alarm when server CPU utilization is too high!"  
        }  
    }  
}
```

```

        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:13.757+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []
    }
}
}

```

Creation of a composite alarm

```

{
    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:22Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:22.289+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR ALARM(TotalNetworkTrafficTooHigh)",
            "alarmName": "ServiceAggregatedAlarm",
            "description": "Aggregated monitor for instance",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:22.289+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}

```

Update of a metric alarm

```

{
    "version": "0",
    "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "operation": "update",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:13.757+0000"
        },
        "configuration": {

```

```

    "evaluationPeriods": 1,
    "threshold": 80,
    "comparisonOperator": "GreaterThanOrEqualToThreshold",
    "treatMissingData": "ignore",
    "metrics": [
        {
            "id": "86bfa85f-b14c-ebf7-8916-7da014ce23c0",
            "metricStat": {
                "metric": {
                    "namespace": "AWS/EC2",
                    "name": "CPUUtilization",
                    "dimensions": {
                        "InstanceId": "i-12345678901234567"
                    }
                },
                "period": 300,
                "stat": "Average"
            },
            "returnData": true
        }
    ],
    "alarmName": "ServerCpuTooHigh",
    "description": "Goes into alarm when server CPU utilization is too high!",
    "actionsEnabled": true,
    "timestamp": "2022-03-03T17:06:34.267+0000",
    "okActions": [],
    "alarmActions": [],
    "insufficientDataActions": []
},
"previousConfiguration": {
    "evaluationPeriods": 1,
    "threshold": 70,
    "comparisonOperator": "GreaterThanOrEqualToThreshold",
    "treatMissingData": "ignore",
    "metrics": [
        {
            "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",
            "metricStat": {
                "metric": {
                    "namespace": "AWS/EC2",
                    "name": "CPUUtilization",
                    "dimensions": {
                        "InstanceId": "i-12345678901234567"
                    }
                },
                "period": 300,
                "stat": "Average"
            },
            "returnData": true
        }
    ],
    "alarmName": "ServerCpuTooHigh",
    "description": "Goes into alarm when server CPU utilization is too high!",
    "actionsEnabled": true,
    "timestamp": "2022-03-03T17:06:13.757+0000",
    "okActions": [],
    "alarmActions": [],
    "insufficientDataActions": []
}
}
}

```

Deletion of a metric math alarm

```
{
}
```

```

"version": "0",
"id": "f171d220-9e1c-c252-5042-2677347a83ed",
"detail-type": "CloudWatch Alarm Configuration Change",
"source": "aws.cloudwatch",
"account": "123456789012",
"time": "2022-03-03T17:07:13Z",
"region": "us-east-1",
"resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
],
"detail": {
    "alarmName": "TotalNetworkTrafficTooHigh",
    "operation": "delete",
    "state": {
        "value": "INSUFFICIENT_DATA",
        "timestamp": "2022-03-03T17:06:17.672+0000"
    },
    "configuration": {
        "evaluationPeriods": 1,
        "threshold": 10000,
        "comparisonOperator": "GreaterThanOrEqualToThreshold",
        "treatMissingData": "ignore",
        "metrics": [
            {
                "id": "m1",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "NetworkIn",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Maximum"
                },
                "returnData": false
            },
            {
                "id": "m2",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "NetworkOut",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Maximum"
                },
                "returnData": false
            },
            {
                "id": "e1",
                "expression": "SUM(METRICS())",
                "label": "Total Network Traffic",
                "returnData": true
            }
        ],
        "alarmName": "TotalNetworkTrafficTooHigh",
        "description": "Goes into alarm if total network traffic exceeds 10Gb",
        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:17.672+0000",
        "okActions": [],
        "alarmActions": []
    }
}

```

```
        "insufficientDataActions": []  
    }  
}
```

Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format

The CloudWatch embedded metric format enables you to ingest complex high-cardinality application data in the form of logs and to generate actionable metrics from them. You can embed custom metrics alongside detailed log event data, and CloudWatch automatically extracts the custom metrics so that you can visualize and alarm on them, for real-time incident detection. Additionally, the detailed log events associated with the extracted metrics can be queried using CloudWatch Logs Insights to provide deep insights into the root causes of operational events.

Embedded metric format helps you to generate actionable custom metrics from ephemeral resources such as Lambda functions and containers. By using the embedded metric format to send logs from these ephemeral resources, you can now easily create custom metrics without having to instrument or maintain separate code, while gaining powerful analytical capabilities on your log data.

When using the embedded metric format, you can generate your logs using a client library—for more information, see [Using the client libraries to generate embedded metric format logs \(p. 762\)](#). Alternatively, you can manually construct the logs and submit them using the PutLogEvents API or the CloudWatch agent.

Charges are incurred for logs ingestion and archival, and custom metrics that are generated. For more information, see [Amazon CloudWatch Pricing](#).

Note

Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see [Dimensions](#).

Topics

- [Generating logs using the embedded metric format \(p. 761\)](#)
- [Viewing your metrics and logs in the console \(p. 774\)](#)

Generating logs using the embedded metric format

You can generate embedded metric format logs with the following methods:

- Generate and send the logs by using the open-sourced client libraries
- Manually generate the logs, and then use the CloudWatch agent or the PutLogEvents API to send the logs

If you use one of the manual methods, the logs must follow the defined JSON format.

Topics

- [Using the client libraries to generate embedded metric format logs \(p. 762\)](#)
- [Using the embedded metric format with AWS Distro for OpenTelemetry \(p. 762\)](#)
- [Manually generating embedded metric format logs \(p. 762\)](#)

Using the client libraries to generate embedded metric format logs

Amazon provides open-sourced client libraries, which you can use to create embedded metric format logs. Currently those libraries are available for the languages in the following list. Support for other languages is planned.

The libraries and the instructions for how to use them are located on Github. Use the following links.

- [Node.js](#)
- [Python](#)
- [Java](#)
- [C#](#)

Using the embedded metric format with AWS Distro for OpenTelemetry

You can use the embedded metric format as a part of the OpenTelemetry project. OpenTelemetry is an open-source initiative that removes boundaries and restrictions between vendor-specific formats for tracing, logs, and metrics by offering a single set of specifications and APIs. For more information, see [OpenTelemetry](#).

Using embedded metric format with OpenTelemetry requires two components: an OpenTelemetry-compliant data source, and the AWS Distro for OpenTelemetry Collector enabled for use with CloudWatch embedded metric format logs.

We have preconfigured redistributions of the OpenTelemetry components, maintained by AWS, to make onboarding as easy as possible. For more information about using OpenTelemetry with embedded metric format, in addition to other AWS services, see [AWS Distro for OpenTelemetry](#).

For additional information regarding language support and usage, see [AWS Observability on Github](#).

Manually generating embedded metric format logs

To send embedded metric format logs that you have manually created, you can use the PutLogEvents API or the CloudWatch agent.

If you use either of these methods, you must follow the embedded metric format specification.

Topics

- [Specification: Embedded metric format \(p. 763\)](#)
- [Using the PutLogEvents API to send manually-created embedded metric format logs \(p. 768\)](#)
- [Using the CloudWatch agent to send embedded metric format logs \(p. 770\)](#)

Specification: Embedded metric format

The CloudWatch embedded metric format is a JSON specification used to instruct CloudWatch Logs to automatically extract metric values embedded in structured log events. You can use CloudWatch to graph and create alarms on the extracted metric values.

Embedded metric format specification conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this format specification are to be interpreted as described in [Key Words RFC2119](#).

The terms "JSON", "JSON text", "JSON value", "member", "element", "object", "array", "number", "string", "boolean", "true", "false", and "null" in this format specification are to be interpreted as defined in [JavaScript Object Notation RFC8259](#).

Embedded metric format specification PutLogEvents request format

Clients MUST use the following log format header when sending an embedded metric format document using the CloudWatch Logs [PutLogEvents API](#):

```
x-amzn-logs-format: json/emf
```

On Lambda, you do not need to set this header yourself. Writing JSON to standard out in the embedded metric format is sufficient. While Lambda may prepend log events with metadata such as timestamp and request id, a valid embedded metric format document after this metadata is considered valid.

Embedded metric format document structure

This section describes the structure of an embedded metric format document, which is identified by the log-format header `x-amzn-logs-format: json/emf`. Embedded metric format documents are defined in [JavaScript Object Notation RFC8259](#).

Unless otherwise noted, objects defined by this specification MUST NOT contain any additional members. Members not recognized by this specification MUST be ignored. Members defined in this specification are case-sensitive.

The embedded metric format is subject to the same limits as standard CloudWatch Logs events and are limited to a maximum size of 256 KB.

Root node

The LogEvent message MUST be a valid JSON object with no additional data at the beginning or end of the LogEvent message string. For more information about the LogEvent structure, see [InputLogEvent](#).

Embedded metric format documents MUST contain the following top-level member on the root node. This is a [Metadata object \(p. 764\)](#) object.

```
{
  "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

The root node MUST contain all [??? \(p. 765\)](#) members defined by the references in the [MetricDirective object \(p. 764\)](#).

The root node MAY contain any other members that are not included in the above requirements. The values of these members MUST be valid JSON types.

Metadata object

The `_aws` member can be used to represent metadata about the payload that informs downstream services how they should process the LogEvent. The value MUST be an object and MUST contain the following members:

- **CloudWatchMetrics**— An array of [MetricDirective object \(p. 764\)](#) used to instruct CloudWatch to extract metrics from the root node of the LogEvent.

```
{  
  "_aws": {  
    "CloudWatchMetrics": [ ... ]  
  }  
}
```

- **Timestamp**— A number representing the time stamp used for metrics extracted from the event. Values MUST be expressed as the number of milliseconds after Jan 1, 1970 00:00:00 UTC.

```
{  
  "_aws": {  
    "Timestamp": 1559748430481  
  }  
}
```

MetricDirective object

The MetricDirective object instructs downstream services that the LogEvent contains metrics that will be extracted and published to CloudWatch. MetricDirectives MUST contain the following members:

- **Namespace**— A string representing the CloudWatch namespace for the metric.
- **Dimensions**— A [DimensionSet array \(p. 764\)](#).
- **Metrics**— An array of [MetricDefinition \(p. 765\)](#) objects. This array MUST NOT contain more than 100 MetricDefinition objects.

DimensionSet array

A DimensionSet is an array of strings containing the dimension keys that will be applied to all metrics in the document. The values within this array MUST also be members on the root-node—referred to as the [Target members \(p. 765\)](#)

A DimensionSet MUST NOT contain more than 9 dimension keys. A DimensionSet MAY be empty.

The target member MUST have a string value. The target member defines a dimension that will be published as part of the metric identity. Every DimensionSet used creates a new metric in CloudWatch. For more information about dimensions, see [Dimension](#) and [Dimensions](#).

```
{  
  "_aws": {  
    "CloudWatchMetrics": [  
      {  
        "Dimensions": [ [ "functionVersion" ] ],  
        ...  
      }  
    ]  
  }  
}
```

```
    },
    "functionVersion": "$LATEST"
}
```

Note

Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see [Dimensions](#).

MetricDefinition object

A MetricDefinition is an object that MUST contain the following member:

- **Name**— A string [Reference values \(p. 765\)](#) to a metric [Target members \(p. 765\)](#). Metric targets MUST be either a numeric value or an array of numeric values.

A MetricDefinition object MAY contain the following member:

- **Unit**— An OPTIONAL string value representing the unit of measure for the corresponding metric. Values SHOULD be valid CloudWatch metric units. For information about valid units, see [MetricDatum](#). If a value is not provided, then a default value of NONE is assumed.

```
{
  "_aws": {
    "CloudWatchMetrics": [
      {
        "Metrics": [
          {
            "Name": "Time",
            "Unit": "Milliseconds"
          }
        ],
        ...
      }
    ],
    "Time": 1
  }
}
```

Reference values

Reference values are string values that reference [Target members \(p. 765\)](#) members on the root node. These references should NOT be confused with the JSON Pointers described in [RFC6901](#). Target values cannot be nested.

Target members

Valid targets MUST be members on the root node and cannot be nested objects. For example, a `_reference_` value of `"A.a"` MUST match the following member:

```
{ "A.a" }
```

It MUST NOT match the nested member:

```
{ "A": { "a" } }
```

Valid values of target members depend on what is referencing them. A metric target MUST be a numeric value or an array of numeric values. Numeric array metric targets MUST NOT have more than 100 members. A dimension target MUST have a string value.

Embedded metric format example and JSON schema

The following is a valid example of embedded metric format.

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds"
          }
        ]
      }
    ],
    "functionVersion": "$LATEST",
    "time": 100,
    "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
  }
}
```

You can use the following schema to validate embedded metric format documents.

```
{
  "type": "object",
  "title": "Root Node",
  "required": [
    "_aws"
  ],
  "properties": {
    "_aws": {
      "$id": "#/properties/_aws",
      "type": "object",
      "title": "Metadata",
      "required": [
        "Timestamp",
        "CloudWatchMetrics"
      ],
      "properties": {
        "Timestamp": {
          "$id": "#/properties/_aws/properties/Timestamp",
          "type": "integer",
          "title": "The Timestamp Schema",
          "examples": [
            1565375354953
          ]
        },
        "CloudWatchMetrics": {
          "$id": "#/properties/_aws/properties/CloudWatchMetrics",
          "type": "array",
          "title": "MetricDirectives",
          "items": {
            "$id": "#/properties/_aws/properties/CloudWatchMetrics/items",
            "type": "object",
            "title": "MetricDirective",
            "required": [
              "MetricName"
            ],
            "properties": {
              "MetricName": {
                "type": "string",
                "title": "The Metric Name"
              },
              "Dimensions": {
                "type": "array",
                "title": "Dimensions"
              },
              "Value": {
                "type": "number",
                "title": "The Value"
              },
              "Unit": {
                "type": "string",
                "title": "The Unit"
              }
            }
          }
        }
      }
    }
  }
}
```

```

        "Namespace",
        "Dimensions",
        "Metrics"
    ],
    "properties": {
        "Namespace": {
            "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Namespace",
                "type": "string",
                "title": "CloudWatch Metrics Namespace",
                "examples": [
                    "MyApp"
                ],
                "pattern": "^(.*)$",
                "minLength": 1,
                "maxLength": 255
            },
        "Dimensions": {
            "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions",
                "type": "array",
                "title": "The Dimensions Schema",
                "minItems": 1,
                "items": {
                    "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions/items",
                        "type": "array",
                        "title": "DimensionSet",
                        "minItems": 0,
                        "maxItems": 9,
                        "items": {
                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items/items",
                                "type": "string",
                                "title": "DimensionReference",
                                "examples": [
                                    "Operation"
                                ],
                                "pattern": "^(.*)$",
                                "minLength": 1,
                                "maxLength": 255
                            }
                        }
                    },
        "Metrics": {
            "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics",
                "type": "array",
                "title": "MetricDefinitions",
                "items": {
                    "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics/items",
                        "type": "object",
                        "title": "MetricDefinition",
                        "required": [
                            "Name"
                        ],
                        "properties": {
                            "Name": {
                                "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/items/properties/Name",
                                    "type": "string",
                                    "title": "MetricName",
                                    "examples": [
                                        "ProcessingLatency"
                                    ],
                                }
                            }
                        }
                    }
    }
}

```

Using the PutLogEvents API to send manually-created embedded metric format logs

You can send embedded metric format logs to CloudWatch Logs using the CloudWatch Logs PutLogEvents API. When calling PutLogEvents, you need to include the following HTTP header to instruct CloudWatch Logs that the metrics should be extracted.

x-amzn-logs-format: json/emf

The following is a full example using the AWS SDK for Java 2.x:

```
package org.example.basicapp;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.InputLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.PutLogEventsRequest;

import java.util.Collections;

public class EmbeddedMetricsExample {
    public static void main(String[] args) {

        final String usage = "To run this example, supply a Region code (eg. us-east-1), log group, and stream name as command line arguments"
                            + "Ex: PutLogEvents <region-id> <log-group-name> <stream-name>";

        if (args.length != 3) {
            System.out.println(usage);
        }
    }
}
```

```

        System.exit(1);
    }

    String regionId = args[0];
    String logGroupName = args[1];
    String logStreamName = args[2];

    CloudWatchLogsClient logsClient =
CloudWatchLogsClient.builder().region(Region.of(regionId)).build();

    // A sequence token is required to put a log event in an existing stream.
    // Look up the stream to find its sequence token.
    String sequenceToken = getNextSequenceToken(logsClient, logGroupName,
logStreamName);

    // Build a JSON log using the EmbeddedMetricFormat.
    long timestamp = System.currentTimeMillis();
    String message = "{" +
        "  \"_aws\": {" +
        "    \"Timestamp\": " + timestamp + "," +
        "    \"CloudWatchMetrics\": [" +
        "      {" +
        "        \"Namespace\": \"MyApp\", " +
        "        \"Dimensions\": [\"Operation\"], [\"Operation\",
\"Cell\"]], " +
        "        \"Metrics\": [{ \"Name\": \"ProcessingLatency\",
\"Unit\": \"Milliseconds\" }]} " +
        "      }" +
        "    ]" +
        "  }, " +
        "  \"Operation\": \"Aggregator\", " +
        "  \"Cell\": \"001\", " +
        "  \"ProcessingLatency\": 100" +
    "}";
    InputLogEvent inputLogEvent = InputLogEvent.builder()
        .message(message)
        .timestamp(timestamp)
        .build();

    // Specify the request parameters.
    PutLogEventsRequest putLogEventsRequest = PutLogEventsRequest.builder()
        .overrideConfiguration(builder ->
            // provide the log-format header of json/emf
            builder.headers(Collections.singletonMap("x-amzn-logs-
format", Collections.singletonList("json/emf"))))
        .logEvents(Collections.singletonList(inputLogEvent))
        .logGroupName(logGroupName)
        .logStreamName(logStreamName)
        .sequenceToken(sequenceToken)
        .build();

    logsClient.putLogEvents(putLogEventsRequest);

    System.out.println("Successfully put CloudWatch log event");
}

private static String getNextSequenceToken(CloudWatchLogsClient logsClient, String
logGroupName, String logStreamName) {
    DescribeLogStreamsRequest logStreamRequest =
DescribeLogStreamsRequest.builder()
        .logGroupName(logGroupName)
        .logStreamNamePrefix(logStreamName)
        .build();

    DescribeLogStreamsResponse describeLogStreamsResponse =
logsClient.describeLogStreams(logStreamRequest);
}

```

```
// Assume that a single stream is returned since a specific stream name was
// specified in the previous request.
return
describeLogStreamsResponse.logStreams().get(0).uploadSequenceToken();
}
```

Using the CloudWatch agent to send embedded metric format logs

To use this method, first install the CloudWatch agent for the services you want to send embedded metric format logs from, and then you can begin sending the events.

The CloudWatch agent must be version 1.230621.0 or later.

Note

You do not need to install the CloudWatch agent to send logs from Lambda functions. Lambda function timeouts are not handled automatically. This means that if your function times out before the metrics get flushed, then the metrics for that invocation will not be captured.

Installing the CloudWatch agent

Install the CloudWatch agent for each service which is to send embedded metric format logs.

Installing the CloudWatch agent on EC2

First, install the CloudWatch agent on the instance. For more information, see [Installing the CloudWatch agent \(p. 483\)](#).

Once you have installed the agent, configure the agent to listen on a UDP or TCP port for the embedded metric format logs. The following is an example of this configuration that listens on the default socket `tcp:25888`. For more information about agent configuration, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Installing the CloudWatch agent on Amazon ECS

The easiest way to deploy the CloudWatch agent on Amazon ECS is to run it as a sidecar, defining it in the same task definition as your application.

Create agent configuration file

Create your CloudWatch agent configuration file locally. In this example, the relative file path will be `amazon-cloudwatch-agent.json`.

For more information about agent configuration, see [Manually create or edit the CloudWatch agent configuration file \(p. 527\)](#).

```
{
  "logs": {
```

```
        "metrics_collected": {
            "emf": { }
        }
    }
}
```

Push configuration to SSM Parameter Store

Enter the following command to push the CloudWatch agent configuration file to the AWS Systems Manager (SSM) Parameter Store.

```
aws ssm put-parameter \
--name "cwagentconfig" \
--type "String" \
--value "`cat amazon-cloudwatch-agent.json`" \
--region "{{region}}"
```

Configure the task definition

Configure your task definition to use the CloudWatch Agent and expose the TCP or UDP port. The sample task definition that you should use depends on your networking mode.

Notice that the webapp specifies the `AWS_EMF_AGENT_ENDPOINT` environment variable. This is used by the library and should point to the endpoint that the agent is listening on. Additionally, the `cwagent` specifies the `CW_CONFIG_CONTENT` as a “valueFrom” parameter that points to the SSM configuration that you created in the previous step.

This section contains one example for bridge mode and one example for host or awsvpc mode. For more examples of how you can configure the CloudWatch agent on Amazon ECS, see the [Github samples repository](#)

The following is an example for bridge mode. When bridge mode networking is enabled, the agent needs to be linked to your application using the `links` parameter and must be addressed using the container name.

```
{
    "containerDefinitions": [
        {
            "name": "webapp",
            "links": [ "cwagent" ],
            "image": "my-org/web-app:latest",
            "memory": 256,
            "cpu": 256,
            "environment": [
                {
                    "name": "AWS_EMF_AGENT_ENDPOINT",
                    "value": "tcp://cwagent:25888"
                }
            ],
            "portMappings": [
                {
                    "protocol": "tcp",
                    "containerPort": 25888
                }
            ],
            "environment": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "cwagentconfig"
                }
            ]
        },
        {
            "name": "cwagent",
            "mountPoints": [],
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "memory": 256,
            "cpu": 256,
            "portMappings": [
                {
                    "protocol": "tcp",
                    "containerPort": 25888
                }
            ],
            "environment": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "cwagentconfig"
                }
            ]
        }
    ]
}
```

```

        }],
    ],
}
}
```

The following is an example for host mode or awsvpc mode. When running on these network modes, the agent can be addressed over localhost.

```

{
  "containerDefinitions": [
    {
      "name": "webapp",
      "image": "my-org/web-app:latest",
      "memory": 256,
      "cpu": 256,
      "environment": [
        {
          "name": "AWS_EMF_AGENT_ENDPOINT",
          "value": "tcp://127.0.0.1:25888"
        }
      ],
      {
        "name": "cwagent",
        "mountPoints": [],
        "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
        "memory": 256,
        "cpu": 256,
        "portMappings": [
          {
            "protocol": "tcp",
            "containerPort": 25888
          }
        ],
        "environment": [
          {
            "name": "CW_CONFIG_CONTENT",
            "valueFrom": "cwagentconfig"
          }
        ],
      }
    ],
  }
}
```

Note

In awsvpc mode, you must either give a public IP address to the VPC (Fargate only), set up a NAT gateway, or set up a CloudWatch Logs VPC endpoint. For more information about setting up a NAT, see [NAT Gateways](#). For more information about setting up a CloudWatch Logs VPC endpoint, see [Using CloudWatch Logs with Interface VPC Endpoints](#).

The following is an example of how to assign a public IP address to a task that uses the Fargate launch type.

```

aws ecs run-task \
--cluster {{cluster-name}} \
--task-definition cwagent-fargate \
--region {{region}} \
--launch-type FARGATE \
--network-configuration
  "awsvpcConfiguration={subnets=[{{subnetId}}],securityGroups=[{{sgId}}],assignPublicIp=ENABLED}"
```

Ensure permissions

Ensure the IAM role executing your tasks has permission to read from the SSM Parameter Store. You can add this permission by attaching the **AmazonSSMReadOnlyAccess** policy. To do so, enter the following command.

```

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess \
```

```
--role-name CWAgentECSExecutionRole
```

Installing the CloudWatch agent on Amazon EKS

Parts of this process can be skipped if you have already installed CloudWatch Container Insights on this cluster.

Permissions

If you have not already installed Container Insights, then first ensure that your Amazon EKS nodes have the appropriate IAM permissions. They should have the **CloudWatchAgentServerPolicy** attached. For more information, see [Verify prerequisites \(p. 320\)](#).

Create ConfigMap

Create a ConfigMap for the agent. The ConfigMap also tells the agent to listen on a TCP or UDP port. Use the following ConfigMap.

```
# cwagent-emf-configmap.yaml
apiVersion: v1
data:
  # Any changes here must not break the JSON format
  cwagentconfig.json: |
    {
      "agent": {
        "omit_hostname": true
      },
      "logs": {
        "metrics_collected": {
          "emf": {}
        }
      }
    }
kind: ConfigMap
metadata:
  name: cwagentemfconfig
  namespace: default
```

If you have already installed Container Insights, add the following "emf": {} line to your existing ConfigMap.

Apply the ConfigMap

Enter the following command to apply the ConfigMap.

```
kubectl apply -f cwagent-emf-configmap.yaml
```

Deploy the agent

To deploy the CloudWatch agent as a sidecar, add the agent to your pod definition, as in the following example.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
    # Your container definitions go here
```

```

- name: web-app
  image: my-org/web-app:latest
# CloudWatch Agent configuration
- name: cloudwatch-agent
  image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
  imagePullPolicy: Always
  resources:
    limits:
      cpu: 200m
      memory: 100Mi
    requests:
      cpu: 200m
      memory: 100Mi
  volumeMounts:
    - name: cwagentconfig
      mountPath: /etc/cwagentconfig
  ports:
# this should match the port configured in the ConfigMap
    - protocol: TCP
      hostPort: 25888
      containerPort: 25888
  volumes:
    - name: cwagentconfig
      configMap:
        name: cwagentemfconfig

```

Using the CloudWatch agent to send embedded metric format logs

When you have the CloudWatch agent installed and running, you can send the embedded metric format logs over TCP or UDP. There are two requirements when sending the logs over the agent:

- The logs must contain a `LogGroupName` key that tells the agent which log group to use.
- Each log event must be on a single line. In other words, a log event cannot contain the newline (\n) character.

The log events must also follow the embedded metric format specification. For more information, see [Specification: Embedded metric format \(p. 763\)](#).

The following is an example of sending log events manually from a Linux bash shell. You can instead use the UDP socket interfaces provided by your programming language of choice.

```

echo '{"_aws": {"Timestamp": 1574109732004, "LogGroupName": "Foo", "CloudWatchMetrics": [{"Namespace": "MyApp", "Dimensions": [[{"Operation": ""}]], "Metrics": [{"Name": "ProcessingLatency", "Unit": "Milliseconds"}]]}, "Operation": "Aggregator", "ProcessingLatency": 1000000000000000000}'} > /dev/udp/0.0.0.0/25888

```

Viewing your metrics and logs in the console

After you generate embedded metric format logs that extract metrics, you can use the CloudWatch console to view the metrics. Embedded metrics have the dimensions that you specified when you generated the logs. Also, embedded metrics that you generated using the client libraries have the following default dimensions:

- `ServiceType`
- `ServiceName`
- `LogGroup`

To view metrics that were generated from embedded metric format logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select a namespace that you specified for your embedded metrics when you generated them. If you used the client libraries to generate the metrics and did not specify a namespace, then select **aws-embedded-metrics**. This is the default namespace for embedded metrics generated using the client libraries.
4. Select a metric dimension (for example, **ServiceName**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To sort the table, use the column heading.
 - b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.

Querying logs using CloudWatch Logs Insights

You can query the detailed log events associated with the extracted metrics by using CloudWatch Logs Insights to provide deep insights into the root causes of operational events. One of the benefits of extracting metrics from your logs is that you can filter your logs later by the unique metric (metric name plus unique dimension set) and metric values, to get context on the events that contributed to the aggregated metric value

For example, to get an impacted request id or x-ray trace id, you could run the following query in CloudWatch Logs Insights.

```
filter Latency > 1000 and Operation = "Aggregator"  
| fields RequestId, TraceId
```

You can also perform query-time aggregation on high-cardinality keys, such as finding the customers impacted by an event. The following example illustrates this.

```
filter Latency > 1000 and Operation = "Aggregator"  
| stats count() by CustomerId
```

For more information, see [Analyzing Log Data with CloudWatch Logs Insights](#)

Use CloudWatch RUM

With CloudWatch RUM, you can perform real user monitoring to collect and view client-side data about your web application performance from actual user sessions in near real time. The data that you can visualize and analyze includes page load times, client-side errors, and user behavior. When you view this data, you can see it all aggregated together and also see breakdowns by the browsers and devices that your customers use.

You can use the collected data to quickly identify and debug client-side performance issues. CloudWatch RUM helps you visualize anomalies in your application performance and find relevant debugging data such as error messages, stack traces, and user sessions. You can also use RUM to understand the range of end user impact including the number of users, geolocations, and browsers used.

End user data that you collect for CloudWatch RUM is retained for 30 days and then automatically deleted. If you want to keep the RUM events for a longer time, you can choose to have the app monitor send copies of the events to CloudWatch Logs in your account. Then, you can adjust the retention period for that log group.

To use RUM, you create an *app monitor* and provide some information. RUM generates a JavaScript snippet for you to paste into your application. The snippet pulls in the RUM web client code. The RUM web client captures data from a percentage of your application's user sessions, which is displayed in a pre-built dashboard. You can specify what percentage of user sessions to gather data from.

The RUM web client is open source. For more information, see [CloudWatch RUM web client](#).

Performance considerations

This section discusses the performance considerations of using CloudWatch RUM.

- **Load performance impact**— The CloudWatch RUM web client is loaded into your web application asynchronously from a content delivery network (CDN). It does not block the application's load process. CloudWatch RUM is designed for there to be no detectable impact to the application's load time.
- **Runtime impact**— The RUM web client performs processing to record and dispatch RUM data to the CloudWatch RUM service. Because events are infrequent and the amount of processing is small, CloudWatch RUM is designed for there to be no detectable impact to the application's performance.
- **Network impact**— The RUM web client periodically sends data to the CloudWatch RUM service. Data is dispatched at regular intervals while the application is running, and also immediately before the browser unloads the application. Data sent immediately before the browser unloads the application are sent as beacons, which, are designed to have no detectable impact on the application's unload time.

RUM Pricing

With CloudWatch RUM, you incur charges for every RUM event that CloudWatch RUM receives. Each data item collected using the RUM web client is considered a RUM event. Examples of RUM events include a page view, a JavaScript error, and an HTTP error. You have options for which types of events are collected by each app monitor. You can activate or deactivate options to collect performance telemetry events, JavaScript errors, HTTP errors, and X-Ray traces. For more information about choosing these options, see [Step 2: Create an app monitor \(p. 780\)](#) and [Information collected by the CloudWatch RUM web client \(p. 790\)](#). For more information about pricing, see [Amazon CloudWatch Pricing](#).

Region availability

CloudWatch RUM is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Europe (Frankfurt)
- Europe (Stockholm)
- Europe (Ireland)
- Europe (London)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)

Topics

- [IAM policies to use CloudWatch RUM \(p. 777\)](#)
- [Set up an application to use CloudWatch RUM \(p. 778\)](#)
- [Configuring the CloudWatch RUM web client with CDN installation \(p. 784\)](#)
- [Viewing the CloudWatch RUM dashboard \(p. 785\)](#)
- [CloudWatch metrics that you can collect with CloudWatch RUM \(p. 786\)](#)
- [Data protection and data privacy with CloudWatch RUM \(p. 789\)](#)
- [Information collected by the CloudWatch RUM web client \(p. 790\)](#)
- [Manage your applications that use CloudWatch RUM \(p. 806\)](#)
- [CloudWatch RUM quotas \(p. 807\)](#)
- [Troubleshooting CloudWatch RUM \(p. 807\)](#)

IAM policies to use CloudWatch RUM

To be able to fully manage CloudWatch RUM, you must be signed in as an IAM user or role that has the **AmazonCloudWatchRUMFullAccess** IAM policy. Additionally, you may need other policies or permissions:

- To create an app monitor that creates a new Amazon Cognito identity pool for authorization, you need to have the **Admin** IAM role or the **AdministratorAccess** IAM policy.
- To create an app monitor that sends data to CloudWatch Logs, you must be logged on to an IAM role or policy that has the following permission:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "logs:PutResourcePolicy"  
    ],  
    "Resource": [  
        "*"  
    ]  
}
```

Other users who need to view CloudWatch RUM data but don't need to create CloudWatch RUM resources, can be granted the **AmazonCloudWatchRUMReadOnlyAccess** policy.

Set up an application to use CloudWatch RUM

Use the steps in these sections to set up your application to begin using CloudWatch RUM to collect performance data from real user sessions.

Topics

- [Step 1: Authorize your application to send data to AWS \(p. 778\)](#)
- [Step 2: Create an app monitor \(p. 780\)](#)
- [\(Optional\) Step 3: Manually modify the code snippet to configure the CloudWatch RUM web client \(p. 781\)](#)
- [Step 4: Insert the code snippet into your application \(p. 783\)](#)
- [Step 5: Test your app monitor setup by generating user events \(p. 783\)](#)

Step 1: Authorize your application to send data to AWS

To use CloudWatch RUM, your application must have authorization.

You have three options to set up authorization:

- Let CloudWatch RUM create a new Amazon Cognito identity pool for the application. This method requires the least effort to set up. It's the default option.

The identity pool will contain an unauthenticated identity. This allows the CloudWatch RUM web client to send data to CloudWatch RUM without authenticating the user of the application.

The Amazon Cognito identity pool has an attached IAM role. The Amazon Cognito unauthenticated identity allows the web client to assume the IAM role that is authorized to send data to CloudWatch RUM.

- Use an existing Amazon Cognito identity pool. In this case, you must also modify the IAM role that is attached to the identity pool.
- Use authentication from an existing identity provider that you have already set up. In this case, you must get credentials from the identity provider and your application must forward these credentials to the RUM web client.

The following sections include more details about these options.

CloudWatch RUM creates a new Amazon Cognito identity pool

This is the simplest option to set up, and if you choose this no further setup steps are required. You must have administrative permissions to use this option. For more information, see [IAM policies to use CloudWatch RUM \(p. 777\)](#).

With this option, CloudWatch RUM creates the following resources:

- A new Amazon Cognito identity pool
- An unauthenticated Amazon Cognito identity. This allows the RUM web client to assume an IAM role without authenticating the user of the application.
- The IAM role that the RUM web client will assume. The IAM policy attached to this role allows it to use the `PutRumEvents` API with the app monitor resource. In other words, it allows the RUM web client to send data to RUM.

The RUM web client uses the Amazon Cognito identity to obtain AWS credentials. The AWS credentials are associated with the IAM role. The IAM role is authorized to use `PutRumEvents` with the `AppMonitor` resource.

Amazon Cognito sends the necessary security token to enable your application to send data to CloudWatch RUM. The JavaScript code snippet that CloudWatch RUM generates includes the following lines to enable authentication.

```
{
    identityPoolId: [identity pool id], // e.g., 'us-west-2:EXAMPLE4a-66f6-4114-902a-
EXAMPLEbad7'
    guestRoleArn: [iam role arn] // e.g., 'arn:aws:iam::123456789012:role/Nexus-
Monitor-us-east-1-123456789012_Unauth_5889316876161'
}
);
```

Use an existing Amazon Cognito identity pool

If you choose to use an existing Amazon Cognito identity pool, you specify the identity pool when you add the application to CloudWatch RUM. The pool must support enabling access to unauthenticated identities. You also must add the following permissions to the IAM policy that is attached to the IAM role that is associated with this identity pool.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountid]:appmonitor/[app monitor name]"
        }
    ]
}
```

Amazon Cognito will then send the necessary security token to enable your application to access CloudWatch RUM.

Third-party provider

If you choose to use private authentication from a third-party provider, you must get credentials from the identity provider and forward them to AWS. The best way to do this is by using a *security token vendor*. You can use any security token vendor, including Amazon Cognito with AWS Security Token Service. For more information about AWS STS, see [Welcome to the AWS Security Token Service API Reference](#).

If you want to use Amazon Cognito as the token vendor in this scenario, you can configure Amazon Cognito to work with an authentication provider. For more information, see [Getting Started with Amazon Cognito Identity Pools \(Federated Identities\)](#).

After you configure Amazon Cognito to work with your identity provider, you also need to do the following:

- Create an IAM role with the following permissions. Your application will use this role to access AWS.

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

```

```
        "Effect": "Allow",
        "Action": "rum:PutRumEvents",
        "Resource": "arn:aws:rum:[region]:[accountID]:appmonitor/[app monitor name]"
    }
}
```

- Add the following to your application to have it pass the credentials from your provider to CloudWatch RUM. Insert the line so that it runs after a user has signed in to your application and the application has received the credentials to use to access AWS.

```
cwr('setAwsCredentials', /* Credentials or CredentialProvider */);
```

For more information about credential providers in the AWS JavaScript SDK, see [Setting credentials in a web browser](#) in the v3 developer guide for SDK for JavaScript, [Setting credentials in a web browser](#) in the v2 developer guide for SDK for JavaScript, , and [@aws-sdk/credential-providers](#).

You can also use the SDK for the CloudWatch RUM web client to configure the web client authentication methods. For more information about the web client SDK, see [CloudWatch RUM web client SDK](#).

Step 2: Create an app monitor

To start using CloudWatch RUM with your application, you create an *app monitor*. When the app monitor is created, RUM generates a JavaScript snippet for you to paste into your application. The snippet pulls in the RUM web client code. The RUM web client captures data from a percentage of your application's user sessions and sends it to RUM.

To create an app monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring**, RUM.
3. Choose **Add app monitor**.
4. Enter the information and settings for your application:
 - For **App monitor name**, enter a name to be used to identify this app monitor within the CloudWatch RUM console.
 - For **Application domain**, enter the top-level domain name where your application has administrative authority. This must be in a URL domain format.

Choose **Include sub domains** to have the app monitor also collect data from all subdomains under the top-level domain.
5. For **Configure RUM data collection**, specify whether you want the app monitor to collect each of the following:
 - **Performance telemetry** – Collects information about page load and resource load times
 - **JavaScript errors** – Collects information about unhandled JavaScript errors raised by your application
 - **HTTP errors** – Collects information about HTTP errors thrown by your application

Selecting these options provides more information about your application, but also generates more CloudWatch RUM events and thus incurs more charges.

If you don't select any of these, the app monitor still collects session start events and page IDs so that you can see how many users are using your application, including breakdowns by operating system type and version, browser type and version, device type, and location.

6. Select **Check this option to allow the CloudWatch RUM Web Client to set cookies** if you want to be able to collect user IDs and session IDs from sampled user sessions. The user IDs are randomly generated by RUM. For more information, see [CloudWatch RUM web client cookies \(p. 789\)](#).
7. For **Session samples**, enter the percentage of user sessions that will be used to gather RUM data. The default is 100%. Reducing this number gives you less data, but reduces your charges. For more information about RUM pricing, see [RUM pricing \(p. 776\)](#).
8. End user data that you collect for CloudWatch RUM is retained for 30 days and then deleted. If you want to keep copies of RUM events in CloudWatch Logs and configure how long to retain these copies, choose **Check this option to store your application telemetry data in your CloudWatch Logs account** under **Data storage**. By default, the CloudWatch Logs log group retains the data for 30 days. You can adjust the retention period in the CloudWatch Logs console.
9. For **Authorization**, specify whether to use a new or existing Amazon Cognito identity pool or use a different identity provider. Creating a new identity pool is the simplest option that requires no other setup steps. For more information, see [Step 1: Authorize your application to send data to AWS \(p. 778\)](#).

Creating a new Amazon Cognito identity pool requires administrative permissions. For more information, see [IAM policies to use CloudWatch RUM \(p. 777\)](#).

10. (Optional) By default, when you add the RUM code snippet to your application, the web client injects the JavaScript tag to monitor usage into the HTML code of all pages of your application. To change this, choose **Configure pages** and then choose either **Include only these pages** or **Exclude these pages**. Then, specify the pages to include or exclude. To specify a page to include or exclude, enter its complete URLs. To specify additional pages, choose **Add URL**.
11. To enable AWS X-Ray tracing of the user sessions that are sampled by the app monitor, choose **Active tracing** and select **Trace my service with AWS X-Ray**.

If you select this, `XMLHttpRequest` and `fetch` requests made during user sessions sampled by the app monitor are traced. You can then see traces and segments from these user sessions in the RUM dashboard, the CloudWatch ServiceLens console, and the X-Ray console.

By making additional configuration changes to the CloudWatch RUM web client, you can add an X-Ray trace header to HTTP requests to enable end-to-end tracing of user sessions through to downstream AWS managed services. For more information, see [Enabling X-Ray end-to-end tracing \(p. 782\)](#).

12. (Optional) To add tags to the app monitor, choose **Tags**, **Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

13. Choose **Add app monitor**.
14. After the JavaScript code snippet is created, choose **Copy to clipboard** or **Download**, and then choose **Done**.

(Optional) Step 3: Manually modify the code snippet to configure the CloudWatch RUM web client

You can modify the code snippet before inserting it into your application, to activate or deactivate several options. For more information, see the [CloudWatch RUM web client documentation](#).

There are three configuration options that you should definitely be aware of, as discussed in these sections.

Preventing the collection of resource URLs that might contain personal information

By default, the CloudWatch RUM web client is configured to record the URLs of resources downloaded by the application. These resources include HTML files, images, CSS files, JavaScript files, and so on. For some applications, URLs may contain personally identifiable information (PII).

If this is the case for your application, we strongly recommend that you disable the collection of resource URLs by setting `recordResourceUrl: false` in the code snippet configuration, before inserting it into your application.

Manually recording page views

By default, the web client records page views when the page first loads and when the browser's history API is called. The page ID is `window.location.pathname`. In some cases, the web client's instrumentation will not record the desired page ID. In this case, you must disable the web client's page view automation by using the `disableAutoPageView` configuration, and the application must be instrumented to record page views using the `recordPageView` command.

Enabling X-Ray end-to-end tracing

When you create the app monitor, selecting **Trace my service with AWS X-Ray** enables the tracing of `XMLHttpRequest` and `fetch` requests made during user sessions that are sampled by the app monitor. You can then see traces from these HTTP requests in the CloudWatch RUM dashboard, the CloudWatch ServiceLens console, and the X-Ray console.

By default, these client-side traces are not connected to downstream server-side traces. To connect client-side traces to server-side traces and enable end-to-end tracing, set the `addXRayTraceIdHeader` option to `true` in the web client. This causes the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests.

The following code block shows an example of adding client-side traces. Some configuration options are omitted from this sample for readability.

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(

    'cwr',
    '00000000-0000-0000-0000-000000000000',
    '1.0.0',
    'us-west-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
        enableXRay: true,
        telemetries: [
            'errors',
            'performance',
            [ 'http', { addXRayTraceIdHeader: true } ]
        ]
    }
);
</script>
```

Warning

Configuring the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests can cause cross-origin resource sharing (CORS) to fail or invalidate the request's signature if the request is signed with SigV4. For more information, see the [CloudWatch RUM web client documentation](#). We strongly recommend that you test your application before adding a client-side X-Ray trace header in a production environment.

For more information, see the [CloudWatch RUM web client documentation](#)

Step 4: Insert the code snippet into your application

Next, you insert the code snippet that you created in the previous section into your application.

Warning

The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. Before you insert the code snippet, see [Data protection and data privacy with CloudWatch RUM \(p. 789\)](#).

If you don't have the code snippet that was previously generated, you can find it by following the directions in [How do I find a code snippet that I've already generated? \(p. 806\)](#).

To insert the CloudWatch RUM code snippet into your application

1. Insert the code snippet that you copied or downloaded in the previous section inside the `<head>` element of your application. Insert it before the `<body>` element or any other `<script>` tags.

The following is an example of a generated code snippet:

```
<script>
(function (n, i, v, r, s, c, x, z) {
    x = window.AwsRumClient = {q: [], n: n, i: i, v: v, r: r, c: c};
    window[n] = function (c, p) {
        x.q.push({c: c, p: p});
    };
    z = document.createElement('script');
    z.async = true;
    z.src = s;
    document.head.insertBefore(z, document.getElementsByTagName('script')[0]);
})('cwr',
    '194a1c89-87d8-41a3-9d1b-5c5cd3dafbd0',
    '1.0.0',
    'us-east-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
        sessionSampleRate: 1,
        guestRoleArn: "arn:aws:iam::123456789012:role/RUM-Monitor-us-
east-2-123456789012-5934510917361-Unauth",
        identityPoolId: "us-east-2:c90ef0ac-e3b8-4d1a-b313-7e73cf21443",
        endpoint: "https://dataplane.rum.us-east-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        enableXRay: false
    });
</script>
```

2. If your application is a multipage web application, you must repeat step 1 for each HTML page that you want included in the data collection.

Step 5: Test your app monitor setup by generating user events

After you have inserted the code snippet and your updated application is running, you can test it by manually generating user events. To test this, we recommend that you do the following. This testing incurs standard CloudWatch RUM charges.

- Navigate between pages in your web application.
- Create multiple user sessions, using different browsers and devices.
- Make requests.
- Cause JavaScript errors.

After you have generated some events, view them in the CloudWatch RUM dashboard. For more information, see [Viewing the CloudWatch RUM dashboard \(p. 785\)](#).

Data from user sessions might take up to 15 minutes to appear in the dashboard.

If you don't see data 15 minutes after you generated events in the application, see [Troubleshooting CloudWatch RUM \(p. 807\)](#).

Configuring the CloudWatch RUM web client with CDN installation

Your applications can use the code snippet generated by CloudWatch RUM to install the CloudWatch RUM web client from a content delivery network (CDN). The code snippet sits in the <head> tag of an HTML file and installs the web client by downloading the web client from a CDN, and then configuring the web client for the application it is monitoring. The snippet is a self-executing function which looks similar to the following. In this example, the body of the snippet's function has been omitted for readability.

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(  
'cwr',  
'00000000-0000-0000-0000-000000000000',  
'1.0.0',  
'us-west-2',  
'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',  
{ /* Configuration Options Here */ }  
);
<script>
```

Arguments

The code snippet accepts six arguments:

- A namespace for running commands on the web client, such as 'cwr'
- The ID of the app monitor, such as '00000000-0000-0000-000000000000'
- The application version, such as '1.0.0'
- The AWS Region of the app monitor, such as 'us-west-2'
- The URL of the web client, such as 'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js'
- Application-specific configuration options. For more information, see the following section.

Configuration options

For information about the configuration options available for the CloudWatch RUM web client, see the [CloudWatch RUM web client documentation](#)

Viewing the CloudWatch RUM dashboard

CloudWatch RUM helps you collect data from user sessions about your application's performance, including page load times, Apdex score, browsers and devices used, geolocation of user sessions, and sessions with errors. All of this information is displayed in a dashboard.

To view the RUM dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, RUM**.

The **Overview** tab displays information collected by one of the app monitors that you have created.

In each tab, you can pause on a chart or graph to see additional details.

The top two rows of panes display the following:

- Number of page loads
- Average page load speed and time
- *Apdex score*
- Sessions with errors
- Status of any alarms associated with the app monitor

The application performance index (Apdex) score indicates end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application. For more information about Apdex scores, see [How CloudWatch RUM sets Apdex scores \(p. 786\)](#).

Several of these panes include links that you can use to further examine the data. Choosing any of these links displays a detailed view with **Performance, Errors & Sessions, Browsers & Devices**, and **User Journey** tabs at the top of the display. The views in each of these tabs include filters you can use to narrow the displayed results down to specific browsers, devices, and countries.

- The **Performance** tab displays page performance information including load times, session information, request information, web vitals, and Apdex score. This view includes controls to toggle the view between focusing on **Page loads, Requests, and Location**.

For more information about Apdex scores and thresholds, see [How CloudWatch RUM sets Apdex scores \(p. 786\)](#).

- The **Errors & Sessions** tab displays page error information including the error message most frequently seen by users, devices and browsers with the most errors, and session error information. This view includes controls to toggle the view between focusing on **Errors, and Sessions**.

Near the bottom of the **Errors** view is a list view of errors. To see more details about one, choose the button at the left of that row in the list, and choose **View details**.

Near the bottom of the **Sessions** view is a list view of sessions. To see more details about one, choose the button at the left of that row in the list, and choose **View session events**.

- The **Browsers & Devices** tab displays information such as the performance and usage of different browsers and devices to access your application. This view includes controls to toggle the view between focusing on **Browsers, \ and Devices**.

If you narrow the scope to a single browser, you see the data broken down by browser version.

- The **User Journey** tab displays the paths that your customers use to navigate your application. You can see where your customers enter your application and what page they exit your application from. You can also see the paths that they take and the percentage of customers that follow those paths. You can pause on a node to get more details about that page. You can choose a single path to highlight the connections for easier viewing.
3. (Optional) On any of the four tabs, you can choose the **Pages** button and select a page from the list. This narrows down the displayed data to a single page of your application. You can also mark pages in the list as favorites.

How CloudWatch RUM sets Apdex scores

Apdex (Application Performance Index) is an open standard that defines a method to report, benchmark, and rate application response time. An Apdex score helps you understand and identify the impact on application performance over time.

The Apdex score indicates the end users' level of satisfaction Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application.

Each individual Apdex score falls into one of three thresholds. Based on the Apdex threshold and actual application response time, there are three kinds of performance, as follows:

- **Satisfied**— The actual application response time is less than or equal to the Apdex threshold. For CloudWatch RUM, this threshold is 2000 ms or less.
- **Tolerable**— The actual application response time is greater than the Apdex threshold, but less than or equal to four times the Apdex threshold. For CloudWatch RUM, this range is 2000–8000 ms.
- **Frustrating**— The actual application response time is greater than four times the Apdex threshold. For CloudWatch RUM, this range is over 8000 ms.

The total 0-1 Apdex score is calculated using the following formula:

`(positive scores + tolerable scores/2)/total scores * 100`

CloudWatch metrics that you can collect with CloudWatch RUM

The following table lists the metrics that you can collect with CloudWatch RUM. You can see these metrics in the CloudWatch console. For more information, see [Viewing available metrics \(p. 71\)](#).

These metrics are published in the metric namespace named `AWS/RUM`. All of the following metrics are published with an `application_name` dimension. The value of this dimension is the name of the app monitor. Some metrics are also published with additional dimensions, as listed in the table.

Metric	Unit	Description
<code>HttpStatusCount</code>	Count	The count of HTTP responses in the application, by their response status code. Additional dimensions:

Metric	Unit	Description
		<ul style="list-style-type: none"> • <code>event_details.response.status</code> is the response status code, such as 200, 400, 404, and so on. • <code>event_type</code> The type of event. Currently, the only possible value for this dimension is <code>http</code>.
JsErrorCount	Count	The count of JavaScript error events ingested.
NavigationFrustratedCount	Count	The count of navigation events with a duration higher than the frustrating threshold, which is 8000ms. The duration of navigation events is tracked in the <code>PerformanceNavigationDuration</code> metric.
NavigationSatisfiedCount	Count	The count of navigation events with a duration that is less than the Apdex objective, which is 2000ms. The duration of navigation events is tracked in the <code>PerformanceNavigationDuration</code> metric.
NavigationToleratedCount	Count	The count of navigation events with a duration between 2000ms and 8000ms. The duration of navigation events is tracked in the <code>PerformanceNavigationDuration</code> metric.

Metric	Unit	Description
PerformanceResourceDuration	Milliseconds	<p>The duration of a resource event.</p> <p>Additional dimensions:</p> <ul style="list-style-type: none"> • <code>event_details.file.type</code> is the file type of the resource event, such as a stylesheet, document, image, script, or font. • <code>event_type</code> The type of event. Currently, the only possible value for this dimension is <code>resource</code>.
PerformanceNavigationDuration	Milliseconds	The duration of a navigation event.
RumEventPayloadSize	Bytes	The size of every event ingested by CloudWatch RUM. You can also use the <code>SampleCount</code> statistic for this metric to monitor the number of events that an app monitor is ingesting.
SessionCount	Count	The count of session start events ingested by the app monitor. In other words, the number of new sessions started.
WebVitalsCumulativeLayoutShift	None	Tracks the value of the cumulative layout shift events.
WebVitalsFirstInputDelay	Milliseconds	Tracks the value of the first input delay events.
WebVitalsLargestContentfulPaint	Milliseconds	Tracks the value of the largest contentful paint events.

Data protection and data privacy with CloudWatch RUM

The AWS [shared responsibility model](#) applies to data protection and data privacy in Amazon CloudWatch RUM. 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. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see [The AWS Shared Responsibility Model and GDPR blog post](#) on the AWS Security Blog. For more resources about complying with GDPR requirements, see the [General Data Protection Regulation \(GDPR\) Center](#).

Amazon CloudWatch RUM generates a code snippet for you to embed into your website or web application code, based on your input of end user data that you want to collect. The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. The use of cookies (or similar technologies) is subject to data privacy regulations in certain jurisdictions. Before using Amazon CloudWatch RUM, we strongly recommend that you assess your compliance obligations under applicable law, including any applicable legal requirements to provide legally adequate privacy notices and obtain any necessary consents for the use of cookies and the processing (including collection) of end user data. For more information about how the web client uses cookies and what end-user data the web client collects, see [Information collected by the CloudWatch RUM web client \(p. 790\)](#) and [CloudWatch RUM web client cookies \(p. 789\)](#).

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Amazon CloudWatch RUM or other services might be included in diagnostic logs.

CloudWatch RUM web client cookies

The CloudWatch RUM web client collects certain data about user sessions by default. You can choose to enable cookies to have the web client also collect a user ID and session ID for each session. The user ID is randomly generated by RUM.

If these cookies are enabled, RUM is able to display the following types of data when you view the RUM dashboard for this app monitor.

- Aggregated data based on user IDs, such as number of unique users and the number of different users who experienced an error.
- Aggregated data based on session IDs, such as number of sessions and the number of sessions that experienced an error.
- The *user journey*, which is the sequence of pages that each sampled user session includes.

Important

If you do not enable these cookies, the web client still records certain information about end user sessions such as browser type/version, operating system type/version, device type, and so on. These are collected to provide aggregated page-specific insights, such as web vitals, page views, and pages that experienced errors. For more information about the data recorded, see [Information collected by the CloudWatch RUM web client \(p. 790\)](#).

Information collected by the CloudWatch RUM web client

This section documents the **PutRumEvents** schema, which defines the structure of the data that you can collect from user sessions using CloudWatch RUM.

A **PutRumEvents** request sends a data structure with the following fields to CloudWatch RUM.

- The ID of this batch of RUM events
- App monitor details, which includes the following:
 - App monitor ID
 - Monitored application version
- User details, which includes the following. **This is collected only if the app monitor has cookies enabled.**
 - A user ID generated by the web client
 - Session ID
- The array of [RUM events \(p. 790\)](#) in this batch.

RUM event schema

The structure of each RUM event includes the following fields.

- The ID of the event
- A timestamp
- The event type
- The user agent
- [Metadata \(p. 790\)](#)
- [RUM event details \(p. 791\)](#)

RUM event metadata

The metadata includes page metadata, user agent metadata, geolocation metadata, and domain metadata.

Page metadata

The page metadata includes the following:

- Page ID
- Page title
- Parent page ID. **This is collected only if the app monitor has cookies enabled.**
- Interaction depth **This is collected only if the app monitor has cookies enabled.**

User agent metadata

The user agent metadata includes the following:

- Browser language

- Browser name
- Browser version
- Operating system name
- Operating system version
- Device type
- Platform type

Geolocation metadata

The geolocation metadata includes the following:

- Country code
- Subdivision code

Domain metadata

The domain metadata includes the URL domain.

RUM event details

The details of an event follow one of the following type of schemas, depending on the event type.

Session start event

This event contains no fields. **This is collected only if the app monitor has cookies enabled.**

Page view schema

A **Page view** event contains the following properties. You can deactivate page view collection by configuring the web client. For more information, see the [CloudWatch RUM web client documentation](#).

Name	Type	Description
Page ID	String	An ID that uniquely represents this page within the application. By default, this is the URL path.
Parent page ID	String	The ID of the page that the user was on when they navigated to the current page. This is collected only if the app monitor has cookies enabled.
Interaction depth	String	This is collected only if the app monitor has cookies enabled.

JavaScript error schema

JavaScript error events generated by the agent contain the following properties. The web client collects these events only if you selected to collect the errors telemetry.

Name	Type	Description
Error type	String	The error's name, if one exists. For more information, see Error.prototype.name .

Name	Type	Description
		Some browsers might not support error types.
Error message	String	The error's message. For more information, see Error.prototype.message . If the error field does not exist, this is the message of the error event. For more information, see ErrorEvent . Error messages might not be consistent across different browsers.
Stack trace	String	The error's stack trace, if one exists, truncated to 150 characters. For more information, see Error.prototype.stack . Some browsers might not support stack traces.

DOM event schema

Document object model (DOM) events generated by the agent contain the following properties. These events are not collected by default. They are collected only if you activate the interactions telemetry. For more information, see the [CloudWatch RUM web client documentation](#).

Name	Type	Description
Event	String	The type of DOM event, such as click, scroll, or hover. For more information, see Event reference .
Element ID	String	If the element that generated the event has an ID, this property stores that ID. For more information, see Element.id . Otherwise, this property stores the tag name. For more information, see Element.tagName .

Navigation event schema

Navigation events are collected only if the app monitor has performance telemetry activated.

Navigation events use [Navigation timing Level 1](#) and [Navigation timing Level 2](#) APIs. Level 2 APIs are not supported on all browsers, so these newer fields are optional.

Note

Timestamp metrics are based on [DOMHighResTimestamp](#). With Level 2 APIs, all timings are by default relative to the `startTime`. But for Level 1, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Navigation events contain the following properties.

Name	Type	Description	Notes
initiatorType	String	Represents the type of resource that initiated the performance event.	Value: "navigation" Level 1: "navigation"

Name	Type	Description	Notes
			Level 2: entryData.initiatorType
navigationType	String	<p>Represents the type of navigation.</p> <p>This attribute is not required.</p>	<p>Value: The value must be one of the following:</p> <ul style="list-style-type: none"> • <code>navigate</code> is a navigation started by choosing a link, entering a URL in a browser's address bar, form submission, or initializing through a script operation other than <code>reload</code> or <code>back_forward</code>. • <code>reload</code> is a navigation through the browser's reload operation or <code>location.reload()</code>. • <code>back_forward</code> is a navigation through the browser's history traversal operation. • <code>prerender</code> is a navigation initiated by a prerender hint. For more information, see Prerender.

Name	Type	Description	Notes
startTime	Number	Indicates when the event is triggered.	Value: 0 Level 1: $\text{entryData.navigationStart} - \text{entryData.navigationStart}$ Level 2: $\text{entryData.startTime}$
unloadEventStartNumber		Indicates the time when the previous document in the window began to unload after the <code>unload</code> event was thrown.	Value: If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0. Level 1: <pre>entryData.unloadEventStart > 0 ? entryData.unloadEventStart - entryData.navigationStart : 0</pre> Level 2: $\text{entryData.unloadEventStart}$

Name	Type	Description	Notes
promptForUnloadNumber	Number	The time taken to unload the document. In other words, the time between <code>unloadEventStart</code> and <code>unloadEventEnd</code> . <code>UnloadEventEnd</code> represents the moment in milliseconds when the unload event handler finishes.	Value: If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0. Level 1: <code>entryData.unloadEventEnd</code> - <code>entryData.unloadEventStart</code> Level 2: <code>entryData.unloadEventEnd</code> - <code>entryData.unloadEventStart</code>
redirectCount	Number	A number representing the number of redirects since the last non-redirect navigation under the current browsing context. This attribute is not required.	Value: If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0. Level 1: Not available Level 2: <code>entryData.redirectCount</code>

Name	Type	Description	Notes
redirectStart	Number	The time when the first HTTP redirect starts.	<p>Value: If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0.</p> <p>Level 1:</p> <pre>entryData.redirectStart > 0 ? entryData.redirectStart - entryData.navigationStart : 0</pre> <p>Level 2: entryData.redirectStart</p>
redirectTime	Number	The time taken for the HTTP redirect. This is the difference between <code>redirectStart</code> and <code>redirectEnd</code> .	<p>Level 1: : entryData.redirectEnd - entryData.redirectStart</p> <p>Level 2: : entryData.redirectEnd - entryData.redirectStart</p>

Name	Type	Description	Notes
workerStart	Number	<p>This is a property of the <code>PerformanceResourceTiming</code> interface. It marks the beginning of worker thread operation.</p> <p>This attribute is not required.</p>	<p>Value: If a Service Worker thread is already running, or immediately before starting the Service Worker thread, this property returns the time immediately before dispatching <code>FetchEvent</code>. It returns 0 if the resource is not intercepted by a Service Worker.</p> <p>Level 1: Not available</p> <p>Level 2: <code>entryData.workerStart</code></p>
workerTime	Number	<p>If the resource is intercepted by a Service Worker, this returns the time required for worker thread operation.</p> <p>This attribute is not required.</p>	<p>Level 1: Not available</p> <p>Level 2:</p> <pre>entryData.workerStart > 0 ? entryData.fetchStart - entryData.workerStart : 0</pre>
fetchStart	Number	The time when the browser is ready to fetch the document using an HTTP request. This is before checking any application cache.	<p>Level 1:</p> <pre>: entryData.fetchStart > 0 ? entryData.fetchStart - entryData.navigationStart : 0</pre> <p>Level 2: <code>entryData.fetchStart</code></p>

Name	Type	Description	Notes
domainLookupStartTime	Number	The time when the domain lookup starts.	<p>Value: If a persistent connection is used or if the information is stored in a cache or local resource, the value will be the same as <code>fetchStart</code>.</p> <p>Level 1:</p> <pre>entryData.domainLookupStart > 0 ? entryData.domainLookupStart - entryData.navigationStart : 0</pre> <p>Level 2: <code>entryData.domainLookupStart</code></p>
dns	Number	The time required for domain lookup.	<p>Value: If the resources and DNS records are cached, the expected value is 0.</p> <p>Level 1: <code>entryData.domainLookupEnd</code> -<code>entryData.domainLookupStart</code></p> <p>Level 2: <code>entryData.domainLookupEnd</code> -<code>entryData.domainLookupStart</code></p>
nextHopProtocolString	String	<p>A string representing the network protocol used to fetch the resource.</p> <p>This attribute is not required.</p>	<p>Level 1: Not available</p> <p>Level 2: <code>entryData.nextHopProtocol</code></p>

Name	Type	Description	Notes
connectStart	Number	The time immediately before the user agent starts establishing the connection to the server to retrieve the document.	<p>Value: If an RFC2616 persistent connection is used, or if the current document is retrieved from relevant application caches or local resources, this attribute returns the value of <code>domainLookupEnd</code>.</p> <p>Level 1:</p> <pre>entryData.connectStart > 0 ? entryData.connectStart - entryData.navigationStart : 0</pre> <p>Level 2: <code>entryData.connectStart</code></p>
connect	Number	Measures the time required to establish the transport connections or to perform SSL authentication. It also includes the blocked time that is taken when there are too many concurrent requests issued by the browser.	<p>Level 1: <code>entryData.connectEnd</code> - <code>entryData.connectStart</code></p> <p>Level 2: <code>entryData.connectEnd</code> - <code>entryData.connectStart</code></p>
secureConnectionStart	Number	If the URL scheme of the current page is "https", this attribute returns the time immediately before the user agent starts the handshake process to secure the current connection. It returns 0 if HTTPS is not used. For more information about URL schemes, see URL representation .	<p>Formula: <code>entryData.secureConnectionStart</code></p>

Name	Type	Description	Notes
tlsTime	Number	The time taken to complete an SSL handshake.	<p>Level 1:</p> <pre>entryData.secureConnection > 0 ? entryData.connectEnd - entryData.secureConnection : 0</pre> <p>Level 2:</p> <pre>entryData.secureConnection > 0 ? entryData.connectEnd - entryData.secureConnection : 0</pre>
requestStart	Number	The time immediately before the user agent starts requesting the resource from the server, or from relevant application caches, or from local resources.	<p>Level 1:</p> <pre>: entryData.requestStart > 0 ? entryData.requestStart - entryData.navigationStart : 0</pre> <p>Level 2: entryData.requestStart</p>
timeToFirstByte	Number	The time taken to receive the first byte of information after a request is made. This time is relative to the <code>startTime</code> .	<p>Level 1: entryData.responseStart -entryData.requestStart</p> <p>Level 2: entryData.responseStart -entryData.requestStart</p>
responseStart	Number	The time immediately after the user agent's HTTP parser receives the first byte of the response from the relevant application caches, or from local resources, or from the server.	<p>Level 1:</p> <pre>entryData.responseStart > 0 ? entryData.responseStart - entryData.navigationStart : 0</pre> <p>Level 2: entryData.responseStart</p>

Name	Type	Description	Notes
responseTime	String	The time taken to receive a complete response in the form of bytes from the relevant application caches, or from local resources, or from the server.	Level 1: <pre>entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0</pre> Level 2: <pre>entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0</pre>
domInteractive	Number	The time when the parser finished its work on the main document, and the HTML DOM is constructed. At this time, its <code>Document.readyState</code> changes to "interactive" and the corresponding <code>readystatechange</code> event is thrown.	Level 1: <pre>entryData.domInteractive > 0 ? entryData.domInteractive - entryData.navigationStart : 0</pre> Level 2: <code>entryData.domInteractive</code>
domContentLoadedEventStart	Number	Represents the time value equal to the time immediately before the user agent fires the <code>DOMContentLoaded</code> event at the current document. The <code>DOMContentLoaded</code> event fires when the initial HTML document has been completely loaded and parsed. At this time, the main HTML document has finished parsing, the browser begins constructing the render tree, and subresources still have to be loaded. This does not wait for style sheets, images, and subframes to finish loading.	Level 1: <pre>entryData.domContentLoadedEventStart > 0 ? entryData.domContentLoadedEventStart - entryData.navigationStart : 0</pre> Level 2: <code>entryData.domContentLoadedEventStart</code>

Name	Type	Description	Notes
domContentLoadedTime	Number	<p>This start and end time of render tree construction is marked by the <code>domContentLoadedEventStart</code> and <code>domContentLoadedEventEnd</code>. It lets CloudWatch RUM track execution. This property is the difference between <code>domContentLoadedStart</code> and <code>domContentLoadedEnd</code>.</p> <p>During this time, DOM and CSSOM are ready. This property waits on script execution, except for asynchronous and dynamically created scripts. If the scripts depend on style sheets, <code>domContentLoaded</code> waits on the style sheets, too. It does not wait on images.</p> <p>Note The actual values of <code>domContentLoadedStart</code> and <code>domContentLoadedEnd</code> approximate to <code>domContentLoaded</code> in Google Chrome's Network panel. It indicates HTML DOM + CSSOM render tree construction time from the beginning of the page loading process. In the case of navigation metrics, the <code>domContentLoaded</code> value represents the difference between start and end values, which is the time required for downloading subresources and render-tree construction only.</p>	<p>Level 2: <code>entryData.domContentLoadedTime</code></p> <p>- <code>entryData.domContentLoadedTime</code></p> <p>Level 2: <code>entryData.domContentLoadedTime</code></p> <p>- <code>entryData.domContentLoadedTime</code></p>
domComplete	Number	The time immediately before the browser sets the current document readiness of the current document to complete. At this point, the loading of subresources, such as images, is complete. This includes the time taken for downloading blocking content such as CSS and synchronous JavaScript. This approximates to <code>loadTime</code> in Google Chrome's Network panel.	<p>Level 1:</p> <pre>entryData.domComplete > 0 ? entryData.domComplete - entryData.navigationStart : 0</pre> <p>Level 2: <code>entryData.domComplete</code></p>
domProcessingTime	Number	The total time between the response and the load event start.	<p>Level 1: <code>entryData.loadEventStart</code></p> <p>- <code>entryData.responseEnd</code></p> <p>Level 2: <code>entryData.loadEventStart</code></p> <p>- <code>entryData.responseEnd</code></p>

Name	Type	Description	Notes
loadEventStart	Number	The time immediately before the load event of the current document is fired.	Level 1: <pre>entryData.loadEventStart > 0 ? entryData.loadEventStart - entryData.navigationStart : 0</pre> Level 2: <code>entryData.loadEventStart</code>
loadEventTime	Number	The difference between <code>loadEventStart</code> and <code>loadEventEnd</code> . Additional functions or logic waiting for this load event will be fired during this time.	Level 1: <code>entryData.loadEventEnd</code> <code>-</code> <code>entryData.loadEventStart</code> Level 2: <code>entryData.loadEventEnd</code> <code>-</code> <code>entryData.loadEventStart</code>
duration	String	Duration is the total page load time. It records the timing for downloading the main page and all of its synchronous subresources, and also for rendering the page. Asynchronous resources such as scripts continue to download later. This is the difference between the <code>loadEventEnd</code> and <code>startTime</code> properties.	Level 1: <code>entryData.loadEventEnd</code> <code>-</code> <code>entryData.navigationStart</code> Level 2: <code>entryData.duration</code>
headerSize	Number	Returns the difference between <code>transferSize</code> and <code>encodedBodySize</code> . This attribute is not required.	Level 1: Not available Level 2: <code>entryData.transferSize</code> <code>-</code> <code>entryData.encodedBodySize</code> Level 2: <code>entryData.transferSize</code> <code>-</code> <code>entryData.encodedBodySize</code>
compressionRatio	Number	The ratio of <code>encodedBodySize</code> and <code>decodedBodySize</code> . The value of <code>encodedBodySize</code> is the compressed size of the resource excluding the HTTP headers. The value of <code>decodedBodySize</code> is the decompressed size of the resource excluding the HTTP headers. This attribute is not required.	Level 1: Not available. Level 2: <pre>entryData.encodedBodySize > 0 ? entryData.decodedBodySize entryData.encodedBodySize : 0</pre>

Name	Type	Description	Notes
<code>navigationTimingLevel</code>	String	The navigation timing API version.	Value: 1 or 2

Resource event schema

Resource events are collected only if the app monitor has performance telemetry activated.

Timestamp metrics are based on [The DOMHighResTimeStamp typedef](#). With Level 2 APIs, by default all timings are relative to the `startTime`. But for Level 1 APIs, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Resource events generated by the agent contain the following properties.

Name	Type	Description	Notes
<code>targetUrl</code>	String	Returns the resource's URL.	Formula: entryData.name
<code>initiatorType</code>	String	Represents the type of resource that initiated the performance resource event.	Value: "resource" Formula: entryData.initiatorType
<code>duration</code>	String	Returns the difference between the <code>responseEnd</code> and <code>startTime</code> properties. This attribute is not required.	Formula: entryData.duration
<code>transferSize</code>	Number	Returns the size (in octets) of the fetched resource, including the response header fields and the response payload body. This attribute is not required.	Formula: entryData.transferSize
<code>fileType</code>	String	Extensions derived from the target URL pattern.	

Largest contentful paint event schema

Largest contentful paint events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
<code>Value</code>	For more information, see Web Vitals .		

First input delay event

First input delay events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description	
Value	For more information, see Web Vitals .	

Cumulative layout shift event

Cumulative layout shift events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description	
Value	For more information, see Web Vitals .	

HTTP event

HTTP events can contain the following properties. It will contain either a `Response` field or an `Error` field, but not both.

These events are collected only if the app monitor has HTTP telemetry activated.

Name	Description
Request	The request field includes the <code>Method</code> field, which can have values such as GET, POST, and so on.
Response	The response field includes the following: <ul style="list-style-type: none"> • Status, such as 2xx, 4xx, or 5xx • Status text
Error	The error field includes the following: <ul style="list-style-type: none"> • Type • Message • File name • Line number • Column number • Stack trace

X-Ray trace event schema

These events are collected only if the app monitor has X-Ray tracing activated.

For information about X-Ray trace event schemas, see [AWS X-Ray segment documents](#).

Manage your applications that use CloudWatch RUM

Use the steps in these sections to manage your applications' use of CloudWatch RUM.

How do I find a code snippet that I've already generated?

To find a CloudWatch RUM code snippet that you've already generated for an application, follow these steps.

To find a code snippet that you've already generated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, RUM**.
3. Choose **List view**.
4. Next to the name of the app monitor, choose **View JavaScript**.
5. In the **JavaScript Snippet** pane, choose **Copy to clipboard**.

Edit your application

To change an app monitor's settings, follow these steps. You can change any settings except the app monitor name.

To edit how your application uses CloudWatch RUM

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, RUM**.
3. Choose **List view**.
4. Choose the button next to the name of the application, and then choose **Actions, Edit**.
5. Change any settings except the application name. For more information about the settings, see [Step 2: Create an app monitor \(p. 780\)](#).
6. When finished, choose **Save**.

Changing the settings changes the code snippet. You must now paste the updated code snippet into your application.

7. After the JavaScript code snippet is created, choose **Copy to clipboard** or **Download**, and then choose **Done**.

To start monitoring with the new settings, you insert the code snippet into your application. Insert the code snippet inside the `<head>` element of your application, before the `<body>` element or any other `<script>` tags.

Stop using CloudWatch RUM or delete an app monitor

To stop using CloudWatch RUM with an application, remove the code snippet that RUM generated from your application's code.

To delete a RUM app monitor, follow these steps.

To delete an app monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, RUM**.
3. Choose **List view**.
4. Choose the button next to the name of the application, and then choose **Actions, Delete**.
5. In the confirmation box, enter **Delete** and then choose **Delete**.
6. If you haven't done so already, delete the CloudWatch RUM code snippet from your application's code.

CloudWatch RUM quotas

CloudWatch RUM has the following quotas.

Resource	Default quota
App monitors	20 per account You can request a quota increase.
RUM events ingestion rate	50 events per second. You can request a quota increase.

Troubleshooting CloudWatch RUM

This section contains tips to help you troubleshoot CloudWatch RUM.

There is no data for my application

First, make sure that the code snippet has been correctly inserted into your application. For more information, see [Step 4: Insert the code snippet into your application \(p. 783\)](#).

If that is not the issue, then maybe there has been no traffic to your application yet. Generate some traffic by accessing your application the same way that a user would.

Data has stopped being recorded for my application

Your application might have been updated and now no longer contains a CloudWatch RUM code snippet. Check your application code.

Another possibility is that someone may have updated the code snippet but then didn't insert the updated snippet into the application. Find the current correct code snippet by following the directions in [How do I find a code snippet that I've already generated? \(p. 806\)](#) and compare it to the code snippet that is pasted into your application.

Perform launches and A/B experiments with CloudWatch Evidently

You can use Amazon CloudWatch Evidently to safely validate new features by serving them to a specified percentage of your users while you roll out the feature. You can monitor the performance of the new feature to help you decide when to ramp up traffic to your users. This helps you reduce risk and identify unintended consequences before you fully launch the feature.

You can also conduct A/B experiments to make feature design decisions based on evidence and data. An experiment can test as many as five variations at once. Evidently collects experiment data and analyzes it using statistical methods. It also provides clear recommendations about which variations perform better. You can test both user-facing features and backend features.

Evidently pricing

Evidently charges your account based on Evidently events and Evidently analysis units. Evidently events include both data events such as clicks and page views, and assignment events that determine the feature variation to serve to a user.

Evidently analysis units are generated from Evidently events, based on rules that you have created in Evidently. Analysis units are the number of rule matches on events. For example, a user click event might produce a single Evidently analysis unit, a click count. Another example is a user checkout event that might produce two Evidently analysis units, checkout value and the number of items in cart. For more information about pricing, see [Amazon CloudWatch Pricing](#).

CloudWatch Evidently is currently available in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Stockholm)

Topics

- [IAM policies to use Evidently \(p. 809\)](#)
- [Create projects, features, launches, and experiments \(p. 810\)](#)
- [Manage features, launches, and experiments \(p. 816\)](#)
- [Adding code to your application \(p. 819\)](#)
- [Project data storage \(p. 820\)](#)

- [View launch results in the dashboard \(p. 822\)](#)
- [View experiment results in the dashboard \(p. 822\)](#)
- [How CloudWatch Evidently collects and stores data \(p. 823\)](#)
- [CloudWatch Evidently quotas \(p. 824\)](#)
- [Tutorial: A/B testing with the Evidently sample application \(p. 824\)](#)

IAM policies to use Evidently

To fully manage CloudWatch Evidently, you must be signed in as an IAM user or role that has the following permissions:

- The **AmazonCloudWatchEvidentlyFullAccess** policy
- The **ResourceGroupsandTagEditorReadOnlyAccess** policy

Additionally, to be able to create a project that stores evaluation events in Amazon S3 or CloudWatch Logs, you need the following permissions:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetBucketPolicy",  
                "s3:PutBucketPolicy",  
                "s3:GetObject",  
                "s3>ListBucket"  
            ],  
            "Resource": "arn:aws:s3:::*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogDelivery",  
                "logs>DeleteLogDelivery",  
                "logs>DescribeResourcePolicies",  
                "logs>PutResourcePolicy"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

Additional permissions for CloudWatch RUM integration

Additionally, if you intend to manage Evidently launches or experiments that integrate with Amazon CloudWatch RUM and use CloudWatch RUM metrics for monitoring, you need the **AmazonCloudWatchRUMFullAccess** policy. To create an IAM role to give the CloudWatch RUM web client permission to send data to CloudWatch RUM, you need the following permissions:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogDelivery",  
                "logs>DeleteLogDelivery",  
                "logs>DescribeResourcePolicies",  
                "logs>PutResourcePolicy"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

```
        "Action": [
            "iam:CreateRole",
            "iam:CreatePolicy",
            "iam:AttachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*",
            "arn:aws:iam::*:policy/service-role/CloudWatchRUMEvidentlyPolicy-*"
        ]
    }
}
```

Permissions for read-only access to Evidently

For other users who need to view Evidently data but don't need to create Evidently resources, you can grant the **AmazonCloudWatchEvidentlyReadOnlyAccess** policy.

Create projects, features, launches, and experiments

To get started with CloudWatch Evidently, for either a feature launch or an A/B experiment, you first create a *project*. A project is a logical grouping of resources. Within the project, you create *features* that have variations that you want to test or launch. You can create a feature either before you create a launch or experiment, or at the same time.

Topics

- [Create a new project \(p. 810\)](#)
- [Add a feature to a project \(p. 811\)](#)
- [Create a launch \(p. 812\)](#)
- [Create an experiment \(p. 814\)](#)

Create a new project

Use these steps to set up a new CloudWatch Evidently project.

To create a new CloudWatch Evidently project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring**, **Evidently**.
3. Choose **Create project**.
4. For **Project name**, enter a name to be used to identify this project within the CloudWatch Evidently console.

You can optionally add a project description.

5. For **Evaluation event storage**, choose whether you want to store the evaluation events that you collect with Evidently. Even if you don't store these events, Evidently aggregates them to create metrics and other experiment data that you can view in the Evidently dashboard. For more information, see [Project data storage \(p. 820\)](#).
6. (Optional) To add tags to this project, choose **Tags**, **Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

7. Choose **Create project**.

Add a feature to a project

A *feature* in CloudWatch Evidently represents a feature that you want to launch or that you want to test variations of.

Before you can add a feature, you must create a project. For more information, see [Create a new project \(p. 810\)](#).

To add a feature to a project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project.
4. Choose **Add feature**.
5. For **Feature name**, enter a name to be used to identify this feature within this project.

You can optionally add a feature description.
6. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. For more information, see [Variation types \(p. 812\)](#).
7. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. This is also the variation that is served to users who are not added to a launch or experiment for this feature.

8. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

You don't need to add the code to your application right now, but you must do so before you start a launch or an experiment.

For more information, see [Adding code to your application \(p. 819\)](#).

9. (Optional) To specify that certain users always see a certain variation, choose **Overrides, Add override**. Then, specify a user by entering their user ID, account ID, or some other identifier in **Identifier**, and specify which variation they should see.

This can be useful for members of your own testing team or other internal users when you want to make sure they see a specific variation. The sessions of users who are assigned overrides do not contribute to launch or experiment metrics.

You can repeat this for as many as 10 users by choosing **Add override** again.

10. (Optional) To add tags to this feature, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

11. Choose **Add feature**.

Variation types

When you create a feature and define the variations, you must select a *variation type*. The possible types are:

- Boolean
- Long integer
- Double precision floating-point number
- String

The variation type sets how the different variations are differentiated in your code. You can use the variation type to simplify the implementation of CloudWatch Evidently and also to simplify the process of modifying the features in your launches and experiments.

For example, if you define a feature with the long integer variation type, the integers that you specify to differentiate the variations can be numbers passed directly into your code. One example might be testing the pixel size of a button. The values for the variation types can be the number of pixels used in each variation. The code for each variation can read the variation type value and use that as the button size. To test a new button size, you can change the number used for the value of the variation, without making any other code changes.

When you set the values for your variation types within a feature, you should avoid assigning the same values to multiple variations, unless you want to do A/A testing to initially try out CloudWatch Evidently, or have other reasons to do so.

Evidently doesn't have native support for JSON as a type, but you can pass in JSON in the String variation type, and parse that JSON in your code.

Create a launch

To expose a new feature or change to a specified percentage of your users, create a launch. You can then monitor key metrics such as page load times and conversions before you roll out the feature to all of your users.

Before you can add a launch, you must have created a project. For more information, see [Create a new project \(p. 810\)](#).

When you add a launch, you can use a feature that you have already created, or create a new feature while you create the launch.

To add a launch to a project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Select the button next to the name of the project and choose **Project actions, Create launch**.
4. For **Launch name**, enter a name to be used to identify this feature within this project.

You can optionally add a description.

5. Choose either **Select from existing features** or **Add new feature**.

If you are using an existing feature, select it under **Feature name**.

If you choose **Add new feature**, do the following:

- a. For **Feature name**, enter a name to be used to identify this feature within this project. You can optionally add a description.
- b. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. For more information, see [Variation types \(p. 812\)](#).
- c. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. If you stop an experiment, this default variation will then be served to all users.

- d. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

You don't need to add the code to your application right now, but you must do so before you start the launch.

For more information, see [Adding code to your application \(p. 819\)](#).

6. For **Launch configuration**, choose whether to start the launch immediately or schedule it to start later.
7. For **Serve**, select the traffic percentage to assign to each variation. You can also choose to exclude variations from being served to users.

The **Traffic summary** shows how much of your overall traffic is available for this launch.

8. If you choose to schedule the launch to start later, you can add multiple steps to the launch. Each step can use different percentages for serving the variations. To do this, choose **Add another step** and then specify the schedule and traffic percentages for the next step. You can include as many as five steps in a launch.
9. If you want to track your feature performance with metrics during the launch, choose **Metrics, Add metric**. You can use either CloudWatch RUM metrics or custom metrics.

To use a custom metric, you can create the metric here using an Amazon EventBridge rule. To create a custom metric, do the following:

- Choose **Custom metrics** and enter a name for the metric.
- Under **Metric rule**, for **Entity ID**, enter the way to identify the entity. This can be a user or session that does an action that causes a metric value to be recorded. An example is `userDetails.userID`.
- For **Value key**, enter the value that is to be tracked to produce the metric.
- Optionally, enter a name for the units for the metric. This unit name is for display purposes only, for use on graphs in the Evidently console.

As you enter those fields, the box shows examples of how to code the EventBridge rule to create the metric. For more information about EventBridge, see [What Is Amazon EventBridge?](#)

To use RUM metrics, you must already have a RUM app monitor set up for your application. For more information, see [Set up an application to use CloudWatch RUM \(p. 778\)](#).

Note

If you use RUM metrics, and the app monitor is not configured to sample 100% of user sessions, then not all of the user sessions that participate in the launch will send metrics to Evidently. To ensure that the launch metrics are accurate, we recommend that the app monitor uses 100% of user sessions for sampling.

10. (Optional) If you create at least one metric for the launch, you can associate an existing CloudWatch alarm with this launch. To do so, choose **Associate CloudWatch alarms**.

When you associate an alarm with a launch, CloudWatch Evidently must add tags to the alarm with the project name and launch name. This is so that CloudWatch Evidently can display the correct alarms in the launch information in the console.

To acknowledge that CloudWatch Evidently will add these tags, choose **Allow Evidently to tag the alarm resource identified below with this launch resource**. Then, choose **Associate alarm** and enter the alarm name.

For information about creating CloudWatch alarms, see [Using Amazon CloudWatch alarms \(p. 130\)](#).

11. (Optional) To add tags to this launch, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

12. Choose **Create launch**.

Create an experiment

Use experiments to test different versions of a feature or website and collect data from real user sessions. This way, you can make choices for your application based on evidence and data.

Before you can add an experiment, you must have created a project. For more information, see [Create a new project \(p. 810\)](#).

When you add an experiment, you can use a feature that you have already created, or create a new feature while you create the experiment.

To add an experiment to a project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Select the button next to the name of the project and choose **Project actions, Create experiment**.
4. For **Experiment name**, enter a name to be used to identify this feature within this project.

You can optionally add a description.

5. Choose either **Select from existing features** or **Add new feature**.

If you are using an existing feature, select it under **Feature name**.

If you choose **Add new feature**, do the following:

- a. For **Feature name**, enter a name to be used to identify this feature within this project. You can also optionally enter a description.
- b. For **Feature variations**, for **Variation type** choose **Boolean, Long, Double, or String**. The type defines which type of value is used for each variation. For more information, see [Variation types \(p. 812\)](#).
- c. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. If you stop

an experiment that uses this feature, the default variation is then served to the percentage of users that were in the experiment previously.

- d. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

You don't need to add the code to your application right now, but you must do so before you start the experiment. For more information, see [Adding code to your application \(p. 819\)](#).

6. For **Audience**, first specify the percentage of the available users whose sessions will be used in the experiment. Then allocate the traffic for the different variations that the experiment uses.

If a launch and an experiment are both running at the same time for the same feature, the audience is first directed to the launch. Then, the percentage of traffic specified for the launch is taken from the overall audience. After that, the percentage that you specify here is the percentage of the remaining audience that is used for the experiment. Any remaining traffic after that is served the default variation.

7. For **Metrics**, choose the metrics to use to evaluate the variations during the experiment. You must use at least one metric for evaluation.

- a. For **Metric source**, choose whether to use CloudWatch RUM metrics or custom metrics.
- b. Enter a name for the metric. For **Goal**, choose **Increase** if you want a higher value for the metric to indicate a better variation. Choose **Decrease** if you want a lower value for the metric to indicate a better variation.
- c. If you are using a custom metric, you can create the metric here using an Amazon EventBridge rule. To create a custom metric, do the following:
 - Under **Metric rule**, for **Entity ID**, enter a way to identify the entity. This can be a user or session that does an action that causes a metric value to be recorded. An example is `userDetails.userID`.
 - For **Value key**, enter the value that is to be tracked to produce the metric.
 - Optionally, enter a name for the units for the metric. This unit name is for display purposes only, for use on graphs in the Evidently console.

You can use RUM metrics only if you have set up RUM to monitor this application. For more information, see [Use CloudWatch RUM \(p. 776\)](#).

Note

If you use RUM metrics, and the app monitor is not configured to sample 100% of user sessions, then not all of the user sessions in the experiment will send metrics to Evidently. To ensure that the experiment metrics are accurate, we recommend that the app monitor uses 100% of user sessions for sampling.

- d. (Optional) To add more metrics to evaluate, choose **Add metric**. You can evaluate as many as three metrics during the experiment.
8. (Optional) To create CloudWatch alarms to use with this experiment, choose **CloudWatch alarms**. The alarms can monitor whether the difference in results between each variation and the default variation is larger than a threshold that you specify. If a variation's performance is worse than the default variation, and the difference is greater than your threshold, it goes into alarm state and notifies you.

Creating an alarm here creates one alarm for each variation that is not the default variation.

If you create an alarm, specify the following:

- For **Metric name**, choose the experiment metric to use for the alarm.

- For **Alarm condition** choose what condition causes the alarm to go into alarm state, when the variation metric values are compared to the default variation metric values. For example, choose **Greater** or **Greater/Equal** if higher numbers indicate for the variation indicates that it is performing poorly. This would be appropriate if the metric is measuring page load time, for example.
- Enter a number for the threshold, which is the percentage difference in performance that will cause the alarm to go into **ALARM** state.
- For **Average over period**, choose how much metric data for each variation is aggregated together before being compared.

You can choose **Add new alarm** again to add more alarms to the experiment.

Next, choose **Set notifications for the alarm** and select or create an Amazon Simple Notification Service topic to send alarm notifications to. For more information, see [Setting up Amazon SNS notifications \(p. 137\)](#),

9. (Optional) To add tags to this experiment, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

10. Choose **Create experiment**.
11. If you haven't already, build the feature variants into your application.
12. Choose **Done**. The experiment does not start until you start it.

After you complete the steps in the following procedure, the experiment starts immediately.

To start an experiment that you have created

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project.
4. Choose the **Experiments** tab.
5. Choose the button next to the name of the experiment, and choose **Actions, Start experiment**.
6. (Optional) To view or modify the experiment settings you made when you created it, choose **Experiment setup**.
7. Choose a time for the experiment to end.
8. Choose **Start experiment**.

The experiment starts immediately.

Manage features, launches, and experiments

Use the procedures in these sections to manage the features, launches, and experiments that you have created.

Topics

- See the current evaluation rules and audience traffic for a feature (p. 817)
- Modify launch traffic (p. 817)
- Modify a launch's future steps (p. 818)

- [Modify experiment traffic \(p. 818\)](#)
- [Stop a launch \(p. 819\)](#)
- [Stop an experiment \(p. 819\)](#)

See the current evaluation rules and audience traffic for a feature

You can use the CloudWatch Evidently console to see how the feature's evaluation rules are allocating the audience traffic among the feature's current launches, experiments, and variations.

To view the audience traffic for a feature

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the feature.
4. Choose the **Features** tab.
5. Choose the name of the feature.

In the **Evaluation rules** tab, you can see the flow of audience traffic for your feature, as follows:

- First, the overrides are evaluated. These specify that certain users are always served a specific variation. The sessions of users who are assigned overrides do not contribute to launch or experiment metrics.
- Next, the remaining traffic is available for the ongoing launch, if there is one. If there is a launch in progress, the table in the **Launches** section displays the launch name and the launch traffic split among the feature variations. On the right side of the **Launches** section, a **Traffic** indicator displays how much of the available audience (after overrides) is allocated to this launch. The rest of the traffic not allocated to the launch flows to the experiment (if any) and then the default variation.
- Next, the remaining traffic is available for the ongoing experiment, if there is one. If there is an experiment in progress, the table in the **Experiments** section displays the experiment name and progress. On the right side of the **Experiments** section, a **Traffic** indicator displays how much of the available audience (after overrides and launches) is allocated to this experiment. The rest of the traffic not allocated to the launch or the experiment is served the default variation of the feature.

Modify launch traffic

You can modify the traffic allocation for a launch at any time, including while the launch is ongoing.

If you have both an ongoing launch and an ongoing experiment for the same feature, any changes to the feature traffic will cause a change in the experiment traffic. This is because the audience available to the experiment is the portion of your total audience that is not already allocated to the launch. Increasing launch traffic will decrease the audience available to the experiment, and decreasing launch traffic or ending the launch will increase the audience available to the experiment.

To modify the traffic allocation for a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launches** tab.

5. Choose the name of the launch.

Choose **Modify launch traffic**.

6. For **Serve**, select the new traffic percentage to assign to each variation. You can also choose to exclude variations from being served to users. As you change these values, you can see the updated effects on your overall feature traffic under **Traffic summary**.

The **Traffic summary** shows how much of your overall traffic is available for this launch, and how much of that available traffic is allocated to this launch.

7. Choose **Modify**.

Modify a launch's future steps

You can modify the configuration of launch steps that haven't happened yet, and also add more steps to a launch.

To modify the steps for a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Application monitoring, Evidently**.

3. Choose the name of the project that contains the launch.

4. Choose the **Launches** tab.

5. Choose the name of the launch.

Choose **Modify launch traffic**.

6. Choose **Schedule launch**.

7. For any steps that have not started yet, you can modify the percentage of the available audience to use in the experiment. You can also modify how their traffic is allocated among the variations.

You can add more steps to the launch by choosing **Add another step**. A launch can have a maximum of five steps.

8. Choose **Modify**.

Modify experiment traffic

You can modify the traffic allocation for an experiment at any time, including while the experiment is ongoing. If you modify the traffic of an ongoing experiment, we recommend that you only increase the traffic allocation, so that you don't introduce bias.

To modify the traffic allocation for an experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Application monitoring, Evidently**.

3. Choose the name of the project that contains the launch.

4. Choose the **Experiments** tab.

5. Choose the name of the launch.

6. Choose **Modify experiment traffic**.

7. Enter a percentage or use the slider to specify how much of the available traffic to allocate to this experiment. The available traffic is the total audience minus the traffic that is allocated to a current launch, if there is one. The traffic that is not allocated to the launch or experiment is served the default variation.

8. Choose **Modify**.

Stop a launch

If you stop an ongoing launch, you will not be able to resume it or restart it. Also, it will not be evaluated as a rule for traffic allocation, and the traffic that was allocated to the launch will instead be available to the feature's experiment, if there is one. Otherwise, all traffic will be served the default variation after the launch is stopped.

To permanently stop a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launch** tab.
5. Choose the button to the left of the name of the launch.
6. Choose **Actions, Cancel launch** or **Actions, Mark as complete**.

Stop an experiment

If you stop an ongoing experiment, you will not be able to resume it or restart it. The portion of traffic that was previously used in the experiment will be served the default variation.

When an experiment is not manually stopped and passes its end date, the traffic does not change. The portion of traffic allocated to the experiment still goes to the experiment. To stop this, and cause the experiment traffic to instead be served the default variation, mark the experiment as complete.

When you stop an experiment, you can choose to cancel it or mark it as complete. If you cancel, it will be shown as **Cancelled** in the list of experiments. If you choose to mark it as complete, it is shown as **Completed**.

To permanently stop an experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the experiment.
4. Choose the **Experiments** tab.
5. Choose the button to the left of the name of the experiment.
6. Choose **Actions, Cancel experiment** or **Actions, Mark as complete**.

Adding code to your application

To work with CloudWatch Evidently, you add code to your application to assign a variation to each user session, and to send metrics to Evidently. When you create variations or custom metrics, the CloudWatch Evidently console provides samples of the code you need to add.

When feature variations are used in a launch or experiment, the application uses the `EvaluateFeature` operation to assign each user session a variation. The assignment of a variation to a user is an *evaluation event*. When you call this operation, you pass in the following:

- **Feature name**— Required. Evidently processes the evaluation according to the feature evaluation rules of the launch or experiment, and selects a variation for the entity.

- **entityId**– Required. Represents a user session.
- **evaluationContext**– Optional. This can includes attributes of the entity to record along with the evaluation event.

To code a custom metric for Evidently, you use the `PutProjectEvents` operation. The following is a simple payload example.

```
{  
    "events": [  
        {  
            "timestamp": {{$timestamp}},  
            "type": "aws.evidently.custom",  
            "data": "{\"details\": {\"pageLoadTime\": 800.0}, \"userDetails\": {\"userId\": \"test-user\"}}"  
        }  
    ]  
}
```

The `entityIdKey` can just be an `entityId` or you can rename it to anything else, such as `userId`. In the actual event, `entityId` can be a username, a session ID, and so on.

```
"metricDefinition":{  
    "name": "noFilter",  
    "entityIdKey": "userDetails.userId", //should be consistent with jsonValue in events  
    "data" fields  
    "valueKey": "details.pageLoadTime"  
},
```

Be sure to call `PutProjectEvents` after the `EvaluateFeature` call, otherwise data is dropped and won't be available on the experiment results page. When you call `PutProjectEvents`, the `entityId` must be the same value as the `entityId` used in the `EvaluateFeature` call, otherwise the custom metric data will be dropped.

Project data storage

Evidently collects two types of events:

- **Evaluation events** are related to which feature variation is assigned to a user session. Evidently uses these events to produce metrics and other experiment and launch data, which you can view in the Evidently console.

You can also choose to store these evaluation events in Amazon CloudWatch Logs or Amazon S3.

- **Custom events** are used to generate metrics from user actions such as clicks and checkouts. Evidently doesn't provide a method for you to store custom events. If you want to save them, you must modify your application code to send them to a storage option outside of Evidently.

Format of evaluation event logs

If you choose to store evaluation events in CloudWatch Logs or Amazon S3, each evaluation event is stored as a log event with the following format:

```
{  
    "event_timestamp": 1642624900215,  
    "event_type": "evaluation",
```

```
"version": "1.0.0",
"project_arn": "arn:aws:evidently:us-east-1:123456789012:project/petfood",
"feature": "petfood-upsell-text",
"variation": "Variation1",
"entity_id": "7",
"entity_attributes": {},
"evaluation_type": "EXPERIMENT_RULE_MATCH",
"treatment": "Variation1",
"experiment": "petfood-experiment-2"
}
```

Here are more details about the preceding evaluation event format:

- The timestamp is in UNIX time with milliseconds
- The variation is the name of the variation of the feature which was assigned to this user session.
- The entity ID is a string.
- Entity attributes are a hash of arbitrary values sent by the client. For example, if the entityId is mapped to blue or green, then you can optionally send userIds, session data, or whatever else that you want from a correlation and data warehouse perspective.

IAM policy and encryption for evaluation event storage in Amazon S3

If you choose to use Amazon S3 to store evaluation events, you must add an IAM policy like the following to allow Evidently to publish logs to the Amazon S3 bucket. This is because Amazon S3 buckets and the objects they contain are private, and they don't allow access to other services by default.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryWrite",
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bucket_name/optional_folder/AWSLogs/account_id/*",
            "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-control"}}
        },
        {
            "Sid": "AWSLogDeliveryCheck",
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": ["s3:GetBucketAcl", "s3>ListBucket"],
            "Resource": "arn:aws:s3:::bucket_name"
        }
    ]
}
```

If you store Evidently data in Amazon S3, you can also choose to encrypt it with Server-Side Encryption with AWS Key Management Service Keys (SSE-KMS). For more information, see [Protecting data using server-side encryption](#).

If you use a customer managed key from AWS KMS, you must add the following to the IAM policy for your key. This allows Evidently to write to the bucket.

```
{
    "Sid": "AllowEvidentlyToUseCustomerManagedKey",
    "Effect": "Allow",
```

```
"Principal": {  
    "Service": [  
        "delivery.logs.amazonaws.com"  
    ]  
},  
"Action": [  
    "kms:Encrypt",  
    "kms:Decrypt",  
    "kms:ReEncrypt*",  
    "kms:GenerateDataKey*",  
    "kms:DescribeKey"  
],  
"Resource": "*"  
}
```

View launch results in the dashboard

You can see the progress and metric results of an experiment while it is ongoing and after it is completed.

To see the progress and results of a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launch** tab.
5. Choose the name of the launch.
6. To see the launch steps and the traffic allocations for each step, choose the **Launch** tab.
7. To see the number of user sessions assigned to each variation over time, and to view the performance metrics for each variation in the launch, choose the **Monitoring** tab.

This view also displays whether any launch alarms have gone into **ALARM** state during the launch.

8. To see the variations, metrics, alarms, and tags for this launch, choose the **Configuration** tab.

View experiment results in the dashboard

You can see the statistical results of an experiment while it is ongoing and after it is completed. Experiment results are available up to 63 days after the start of the experiment. They are not available after that because of CloudWatch data retention policies.

No statistical results are displayed until each variation has at least 100 events.

To see the results of an experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the experiment.
4. Choose the **Experiments** tab.
5. Choose the name of the experiment, and then choose the **Results** tab.
6. By **Variation performance**, there is a control where you can select which experiment statistics to display. If you select more than one statistic, Evidently displays a graph and table for each statistic.

Each graph and table displays the results of the experiment so far.

Each graph can display the following results. You can use the control at the right of the graph to determine which of the following items is displayed:

- The number of user session events recorded for each variation.
- The average value of the metric that is selected at the top of the graph, for each variation.
- The statistical significance of the experiments. This compares the difference for the metric selected at the top of the graph with the default variation and each of the other variations.
- The 95% upper and lower confidence bounds on the difference of the selected metric, between each of the variations and the default variation.

The table displays a row for each variation. For each variation that is not the default, Evidently displays whether it has received enough data to declare the results statistically significant. It also shows whether the variation's improvement in the statistical value has reached a 95% confidence level.

Finally, in the **Result** column, Evidently provides a recommendation about which variation performs best based on this statistic, or whether the results are inconclusive.

How CloudWatch Evidently collects and stores data

Amazon CloudWatch Evidently collects and stores data related to project configurations so that customers can run experiments and launches. The data includes the following:

- Metadata about projects, features, launches, and experiments
- Metric events
- Evaluation data

Resource metadata is stored in Amazon DynamoDB. The data is encrypted at rest by default, using AWS owned keys. These keys are a collection of AWS KMS keys that an AWS service owns and manages for use in multiple AWS accounts. Customers can't view, manage, or audit the use of these keys. Customers are also not required to take action or change programs to protect the keys that encrypt their data.

For more information, see [AWS owned keys](#) in the AWS Key Management Service Developer Guide.

Evidently metric events and evaluation events are delivered directly to customer-owned locations.

Evidently also temporarily creates and stores evaluation logs in Amazon Elastic Container Service hosts before being rotated to Ingestion Hub. These logs are currently retained for a maximum of three seconds and deleted after processing. Data in transit is automatically encrypted with HTTPS. This data will be delivered to customer-owned locations.

You can also choose to store evaluation events in Amazon Simple Storage Service or Amazon CloudWatch Logs. For more information about how you can secure your data in these services, see [Enabling Amazon S3 default bucket encryption](#) and [Encrypting log data in CloudWatch Logs using AWS KMS](#).

Retrieving data

You can retrieve your data using CloudWatch Evidently APIs. To retrieve project data, use [GetProject](#) or [ListProjects](#).

To retrieve feature data, use [GetFeature](#) or [ListFeatures](#).

To retrieve launch data, use [GetLaunch](#) or [ListLaunches](#).

To retrieve experiment data, use [GetExperiment](#), [ListExperiments](#), or [GetExperimentResults](#).

Modifying and deleting data

You can modify and delete your data using CloudWatch Evidently APIs. For project data, use [UpdateProject](#) or [DeleteProject](#).

For feature data, use [UpdateFeature](#) or [DeleteFeature](#).

For launch data, use [UpdateLaunch](#) or [DeleteLaunch](#).

For experiment data, use [UpdateExperiment](#) or [DeleteExperiment](#).

CloudWatch Evidently quotas

CloudWatch Evidently has the following quotas.

Resource	Default quota
Projects	50 per Region per account
Limits per project	100 total features 500 total launches 50 running launches 500 total experiments 50 running experiments
API limits (all limits are per Region)	PutProjectEvents: 50 transactions per second (TPS) EvaluateFeature: 50 TPS BatchEvaluateFeature: 50 TPS Create, Read, Update, Delete (CRUD) APIs: 10 TPS combined across all CRUD APIs

Tutorial: A/B testing with the Evidently sample application

This section provides a tutorial for using Amazon CloudWatch Evidently for A/B testing. This tutorial uses the Evidently sample application, which is a simple react application. The sample app will be configured to either display a `showDiscount` feature or not. When the feature is shown to a user, the price displayed on the shopping website is shown at a 20% discount.

In addition to showing the discount to some users and not to others, in this tutorial you set up Evidently to collect page load time metrics from both variations.

Step 1: Download the sample application

Start by downloading the Evidently sample application.

To download the sample application

1. Download the sample application from the following Amazon S3 bucket:

```
https://evidently-sample-application.s3.us-west-2.amazonaws.com/evidently-sample-shopping-app.zip
```

2. Unzip the package.

Step 2: Add the Evidently endpoint and set up credentials

Next, add the Region and endpoint for Evidently to the `config.js` file in the `src` directory in the sample app package, as in the following example:

```
evidently: {  
    REGION: "us-west-2",  
    ENDPOINT: "https://evidently.us-west-2.amazonaws.com (https://evidently.us-west-2.amazonaws.com/)",  
},
```

You also must make sure that the application has permission to call CloudWatch Evidently.

To grant the sample app permissions to call Evidently

1. Federate to your AWS account.
2. Create an IAM user and attach the **AmazonCloudWatchEvidentlyFullAccess** policy to this user.
3. Make a note of the IAM user's access key id and secret access key, because you will need them in the next step.
4. In the same `config.js` file that you modified earlier in this section, enter the values of the access key ID and the secret access key, as in the following example:

```
credential: {  
    accessKeyId: "Access key ID",  
    secretAccessKey: "Secret key"  
}
```

Important

We use this step to make the sample app as simple as possible for you to try out. We do not recommend that you put your IAM user credential into your actual production application. Instead, we recommend that you use Amazon Cognito for authentication. For more information, see [Integrating Amazon Cognito with web and mobile apps](#).

Step 3: Set up code for the feature evaluation

When you use CloudWatch Evidently to evaluate a feature, you must use the **EvaluateFeature** operation to randomly select a feature variation for each user session. This operation assigns user sessions to each variation of the feature, according to the percentages that you specified in the experiment.

To set up the feature evaluation code for the bookstore demo app

1. Add the client builder in the `src/App.jsx` file so that the sample app can call Evidently.

```

import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
    endpoint,
    region,
) => {
    const credentials = {
        accessKeyId: config.credential.accessKeyId,
        secretAccessKey: config.credential.secretAccessKey
    }
    return new Evidently({
        endpoint,
        region,
        credentials,
    });
};

```

2. Add the following to the `const App` code section to initiate the client.

```

if (client == null) {
    client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
    );
}

```

3. Construct `evaluateFeatureRequest` by adding the following code. This code pre-fills the project name and feature name that we recommend later in this tutorial. You can substitute your own project and feature names, as long as you also specify those project and feature names in the Evidently console.

```

const evaluateFeatureRequest = {
    entityId: id,
    // Input Your feature name
    feature: 'showDiscount',
    // Input Your project name
    project: 'EvidentlySampleApp',
};

```

4. Add the code to call Evidently for feature evaluation. When the request is sent, Evidently randomly assigns the user session to either see the `showDiscount` feature or not.

```

client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
    }
    getPageLoadTime()
})

```

Step 4: Set up code for the experiment metrics

For the custom metric, use Evidently's `PutProjectEvents` API to send metric results to Evidently. The following examples show how to set up the custom metric and send experiment data to Evidently.

Add the following function to calculate the page load time and use `PutProjectEvents` to send the metric values to Evidently. Add the following function to `Home.tsx` and call this function within the `EvaluateFeature` API:

```

const getPageLoadTime = () => {
  const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
  const pageLoadTimeData = `{
    "details": {
      "pageLoadTime": ${timeSpent}
    },
    "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
  }`;
  const putProjectEventsRequest = {
    project: 'EvidentlySampleApp',
    events: [
      {
        timestamp: new Date(),
        type: 'aws.evidently.custom',
        data: JSON.parse(pageLoadTimeData)
      },
    ],
  };
  client.putProjectEvents(putProjectEventsRequest).promise();
}

```

Here is what the `App.js` file should look like after all the editing that you have done since downloading it.

```

import React, { useEffect, useState } from "react";
import { BrowserRouter as Router, Switch } from "react-router-dom";
import AuthProvider from "contexts/auth";
import CommonProvider from "contexts/common";
import ProductsProvider from "contexts/products";
import CartProvider from "contexts/cart";
import CheckoutProvider from "contexts/checkout";
import RouteWrapper from "layouts/RouteWrapper";
import AuthLayout from "layouts/AuthLayout";
import CommonLayout from "layouts/CommonLayout";
import AuthPage from "pages/auth";
import HomePage from "pages/home";
import CheckoutPage from "pages/checkout";
import "assets/scss/style.scss";
import { Spinner } from 'react-bootstrap';

import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
  endpoint,
  region,
) => {
  const credentials = {
    accessKeyId: config.credential.accessKeyId,
    secretAccessKey: config.credential.secretAccessKey
  }
  return new Evidently({
    endpoint,
    region,
    credentials,
  });
};

const App = () => {
  const [isLoading, setIsLoading] = useState(true);
  const [startTime, setStartTime] = useState(new Date());
  const [showDiscount, setShowDiscount] = useState(false);
  let client = null;
  let id = null;

```

```

useEffect(() => {
  id = new Date().getTime().toString();
  setStartTime(new Date());
  if (client == null) {
    client = defaultClientBuilder(
      config.evidently.ENDPOINT,
      config.evidently.REGION,
    );
  }
  const evaluateFeatureRequest = {
    entityId: id,
    // Input Your feature name
    feature: 'showDiscount',
    // Input Your project name
    project: 'EvidentlySampleApp',
  };

  // Launch
  client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
      setShowDiscount(res.value.boolValue);
    }
  });
}

// Experiment
client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
  if(res.value?.boolValue !== undefined) {
    setShowDiscount(res.value.boolValue);
  }
  getPageLoadTime()
})

setIsLoading(false);
},[]);

const getPageLoadTime = () => {
  const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
  const pageLoadTimeData = `{
    "details": {
      "pageLoadTime": ${timeSpent}
    },
    "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
  }`;
  const putProjectEventsRequest = {
    project: 'EvidentlySampleApp',
    events: [
      {
        timestamp: new Date(),
        type: 'aws.evidently.custom',
        data: JSON.parse(pageLoadTimeData)
      },
    ],
  };
  client.putProjectEvents(putProjectEventsRequest).promise();
}
return (
  !isLoading? (
    <AuthProvider>
      <CommonProvider>
        <ProductsProvider>
          <CartProvider>
            <CheckoutProvider>
              <Router>
                <Switch>
                  <RouteWrapper>

```

```
path="/" exact component={() => <HomePage showDiscount={showDiscount}/>} layout={CommonLayout} />
<RouteWrapper path="/checkout" component={CheckoutPage} layout={CommonLayout} />
<RouteWrapper path="/auth" component={AuthPage} layout={AuthLayout} />
</Switch>
</Router>
</CheckoutProvider>
</CartProvider>
</ProductsProvider>
</CommonProvider>
</AuthProvider> ) : (
  <Spinner animation="border" />
)
);
};

export default App;
```

Each time a user visits the sample app, a custom metric is sent to Evidently for analysis. Evidently analyzes each metric and displays results in real time on the Evidently dashboard. The following example shows a metric payload:

```
[ {"timestamp": 1637368646.468, "type": "aws.evidently.custom", "data": "{\"details\": {\"pageLoadTime\":2058.002058},\"userDetails\":{\"userId\":\"1637368644430\",\"sessionId\":\"1637368644430\"}}"} ]
```

Step 5: Create the project, feature, and experiment

Next, you create the project, feature, and experiment in the CloudWatch Evidently console.

To create the project, feature, and experiment for this tutorial

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring**, **Evidently**.
3. Choose **Create project** and fill out the fields. You must use **EvidentlySampleApp** for the project name for the sample to work correctly. For **Evaluation event storage**, choose **Don't store Evaluation events**.

After filling out the fields, choose **Create Project**.

For more details, see [Create a new project \(p. 810\)](#).

4. After the project is created, create a feature in that project. Name the feature **showDiscount**. In this feature, create two variations of the **Boolean** type. Name the first variation **disable** with a value of **False** and name the second variation **enable** with a value of **True**.

For more information about creating a feature, see [Add a feature to a project \(p. 811\)](#).

5. After you have finished creating the feature, create an experiment in the project. Name the experiment **pageLoadTime**.

This experiment will use a custom metric called `pageLoadTime` that measures the page load time of the page being tested. Custom metrics for experiments are created using Amazon EventBridge. For more information about EventBridge, see [What Is Amazon EventBridge?](#).

To create that custom metric, do the following when you create the experiment:

- Under **Metrics**, for **Metric source**, choose **Custom metrics**.
- For **Metric name**, enter `pageLoadTime`.
- For **Goal** choose **Decrease**. This indicates that we want a lower value of this metric to indicate the best variation of the feature.
- For **Metric rule**, enter the following:
 - For **Entity ID**, enter `UserDetails.userId`.
 - For **Value key**, enter `details.pageLoadTime`.
 - For **Units**, enter `ms`.
- Choose **Add metric**.

For **Audiences**, select **100%** so that all users are entered in the experiment. Set up the traffic split between the variations to be 50% each.

Then, choose **Create experiment** to create the experiment. After you create it, it does not start until you tell Evidently to start it.

Step 6: Start the experiment and test CloudWatch Evidently

The final steps are starting the experiment and starting the sample app.

To start the tutorial experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring**, **Evidently**.
3. Choose the **EvidentlySampleApp** project.
4. Choose the **Experiments** tab.
5. Choose the button next to `pageLoadTime` and choose **Actions**, **Start experiment**.
6. Choose a time for the experiment to end.
7. Choose **Start experiment**.

The experiment starts immediately.

Next, start the Evidently sample app with the following command:

```
npm install -f && npm start
```

Once the app has started, you will be assigned to one of the two feature variations being tested. One variation displays "20% discount" and the other doesn't. Keep refreshing the page to see the different variations.

Note

Evidently has sticky evaluations. Feature evaluations are deterministic, meaning for the same `entityId` and feature, a user will receive the same variation assignment. The only time

variation assignments change is when an entity is added to an override or experiment traffic is dialed up.

However, to make the use of the sample app tutorial easy for you, Evidently reassigns the sample app feature evaluation every time that you refresh the page, so that you can experience both variations without having to add overrides.

Troubleshooting

We recommend that you use npm version 6.14.14. If you see any errors about building or starting the sample app and you are using a different version of npm, do the following.

To install npm version 6.14.14

1. Use a browser to connect to <https://nodejs.org/download/release/v14.17.5/>.
2. Download [node-v14.17.5.pkg](#) and run this pkg to install npm.

If you see a webpack not found error, navigate to the evidently-sample-shopping-app folder and try the following:

- a. Delete package-lock.json
- b. Delete yarn-lock.json
- c. Delete node_modules
- d. Delete the webpack dependency from package.json
- e. Run the following:

```
npm install -f && npm
```

AWS usage metrics

CloudWatch collects metrics that track the usage of some AWS resources and APIs. These metrics are published in the AWS/Usage namespace. Usage metrics in CloudWatch allow you to proactively manage usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch anomaly detection, and configuring alarms that alert you when usage approaches a threshold.

Some AWS services integrate these usage metrics with Service Quotas. For these services, you can use CloudWatch to manage your account's use of your service quotas. For more information, see [Visualizing your service quotas and setting alarms \(p. 832\)](#).

Topics

- [Visualizing your service quotas and setting alarms \(p. 832\)](#)
- [AWS API usage metrics \(p. 833\)](#)
- [CloudWatch usage metrics \(p. 838\)](#)

Visualizing your service quotas and setting alarms

For some AWS services, you can use the usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information about service quotas, see [What Is Service Quotas](#) in the *Service Quotas User Guide*.

Currently, the following services integrate their usage metrics with Service Quotas:

- AWS CloudHSM
- [Amazon CloudWatch](#)
- [Amazon CloudWatch Logs](#)
- [Amazon DynamoDB](#)
- [Amazon EC2](#)
- [Amazon Elastic Container Registry](#)
- [AWS Fargate](#)
- [AWS Fault Injection Simulator](#)
- [AWS Interactive Video Service](#)
- [AWS Key Management Service](#)
- [Amazon Kinesis Data Firehose](#)
- [AWS RoboMaker](#)

To visualize a service quota and optionally set an alarm

1. 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 **Usage**, and then choose **By AWS Resource**.

The list of service quota usage metrics appears.

4. Select the check box next to one of the metrics.

The graph displays your current usage of that AWS resource.

5. To add your service quota to the graph, do the following:
 - a. Choose the **Graphed metrics** tab.
 - b. Choose **Math expression, Start with an empty expression**. In the new row, under **Details**, enter **SERVICE_QUOTA(m1)**.

A new line is added to the graph, displaying the service quota for the resource represented in the metric.
6. To see your current usage as a percentage of the quota, add a new expression or change the current **SERVICE_QUOTA** expression. The new expression to use is **m1/SERVICE_QUOTA(m1)*100**.
7. (Optional) To set an alarm that notifies you if you approach the service quota, do the following:
 - a. On the row with **m1/SERVICE_QUOTA(m1)*100**, under **Actions**, choose the alarm icon. It looks like a bell.
 - The alarm creation page appears.
 - b. Under **Conditions**, ensure that **Threshold type** is **Static** and **Whenever Expression1** is set to **Greater**. Under **than**, enter **80**. This creates an alarm that goes into ALARM state when your usage exceeds 80 percent of the quota.
 - c. Choose **Next**.
 - d. On the next page, select an Amazon SNS topic or create a new one, and then choose **Next**. The topic you select is notified when the alarm goes to ALARM state.
 - e. On the next page, enter a name and description for the alarm, and then choose **Next**.
 - f. Choose **Create alarm**.

AWS API usage metrics

Most APIs that support AWS CloudTrail logging also report usage metrics to CloudWatch. API usage metrics in CloudWatch allow you to proactively manage API usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch Anomaly Detection, and configuring alarms that alert when usage approaches a threshold.

The following table lists the services that report API usage metrics to CloudWatch, and the value to use for the **Service** dimension to see the usage metrics from that service.

Service	Value for the Service dimension
AWS Identity and Access Management Access Analyzer	Access Analyzer
AWS Account Management	Account Management
Alexa for Business	A4B
Amazon API Gateway	API Gateway
AWS App Mesh	App Mesh
AWS AppConfig	AWS AppConfig
Amazon AppFlow	AppFlow
Application Discovery Service	Application Discovery Service
Amazon AppStream	AppStream

Service	Value for the Service dimension
AppStream 2.0 Image Builder	Image Builder
Amazon Athena	Athena
AWS Audit Manager	Audit Manager
AWS Backup	Backup
AWS Batch	Batch
Amazon Braket	Braket
AWS Budgets	Budgets
AWS Certificate Manager	Certificate Manager
Amazon Cloud Directory	Cloud Directory
AWS Cloud Map	Cloud Map
AWS CloudFormation	CloudFormation
AWS CloudHSM	CloudHSM
Amazon CloudSearch	CloudSearch
AWS CloudShell	CloudShell
AWS CloudTrail	CloudTrail
Amazon CloudWatch	CloudWatch
Amazon CloudWatch Logs	Logs
Amazon CloudWatch Application Insights	CloudWatch Application Insights
AWS CodeBuild	CodeBuild
AWS CodeCommit	CodeCommit
Amazon CodeGuru Profiler	CodeGuru Profiler
AWS CodePipeline	CodePipeline
AWS CodeStar	CodeStar
AWS CodeStar Notifications	CodeStar Notifications
AWS CodeStar Connections	CodeStar Connections
Amazon Cognito Identity pools	Cognito Identity Pools
Amazon Cognito Sync	Cognito Sync
Amazon Comprehend	Comprehend
Amazon Comprehend Medical	Comprehend Medical
AWS Compute Optimizer	ComputeOptimizer
Amazon Connect	Connect

Service	Value for the Service dimension
Amazon Connect Customer Profiles	Customer Profiles
AWS Cost and Usage Reports	Cost and Usage Report
AWS Cost Explorer	Cost Explorer
AWS Data Exchange	Data Exchange
AWS Data Lifecycle Manager	Data Lifecycle Manager
AWS Database Migration Service	Database Migration Service
AWS DataSync	DataSync
AWS DeepLens	AWS DeepLens
Amazon Detective	Detective
Device Advisor	Device Advisor
AWS Direct Connect	Direct Connect
AWS Directory Service	Directory Service
DynamoDB Accelerator	DynamoDBAccelerator
Amazon EC2	EC2
Amazon Elastic Container Registry	ECR Public
Amazon Elastic Container Service	ECS
Amazon Elastic File System	EFS
Amazon Elastic Kubernetes Service	EKS
AWS Elastic Beanstalk	Elastic Beanstalk
Amazon Elastic Inference	Elastic Inference
Elastic Load Balancing	Elastic Load Balancing
Amazon EMR	EMR Containers
AWS Firewall Manager	Firewall Manager
Amazon FSx	FSx
Amazon GameLift	GameLift
AWS Glue DataBrew	DataBrew
Amazon Managed Grafana	Grafana
AWS IoT Greengrass	Greengrass
AWS Ground Station	Ground Station
AWS Health APIs And Notifications	AWS Health APIs And Notifications
Amazon Interactive Video Service	IVS

Service	Value for the Service dimension
AWS IoT Core	IoT
AWS IoT 1-Click	IoT 1-Click
AWS IoT Events	IoT Events
AWS IoT RoboRunner	IoT RoboRunner
AWS IoT SiteWise	IoT Sitewise
AWS IoT Wireless	IoT Wireless
Amazon Kendra	Kendra
Amazon Keyspaces (for Apache Cassandra)	Keyspaces
Amazon Kinesis Data Analytics	Kinesis Analytics
Amazon Kinesis Data Firehose	Firehose
Kinesis Video Streams	Kinesis Video Streams
AWS Key Management Service	KMS
AWS Lambda	Lambda
AWS Launch Wizard	Launch Wizard
Amazon Lex	Amazon Lex
Amazon Lightsail	Lightsail
Amazon Location Service	Location
Amazon Lookout for Vision	Lookout for Vision
Amazon Machine Learning	Amazon Machine Learning
Amazon Macie	Macie
AWS Managed Services	AWS Managed Services
AWS Marketplace Commerce Analytics	Marketplace Analytics Service
AWS Elemental MediaConnect	MediaConnect
AWS Elemental MediaConvert	MediaConvert
AWS Elemental MediaLive	MediaLive
AWS Elemental MediaStore	MediaStore
AWS Elemental MediaTailor	MediaTailor
AWS Mobile Hub	Mobile Hub
AWS Network Firewall	Network Firewall
AWS OpsWorks	OpsWorks
AWS OpsWorks for Configuration Management	OpsWorks CM

Service	Value for the Service dimension
AWS Outposts	Outposts
AWS Organizations	Organizations
Amazon RDS Performance Insights	Performance Insights
Amazon Pinpoint	Pinpoint
AWS Certificate Manager Private Certificate Authority	Private Certificate Authority
Amazon Managed Service for Prometheus	Prometheus
AWS Proton	Proton
Amazon Quantum Ledger Database (Amazon QLDB)	QLDB
Amazon RDS	RDS
Amazon Redshift	Redshift Data API
Amazon Rekognition	Rekognition
AWS Resource Access Manager	Resource Access Manager
AWS Resource Groups	Resource Groups
AWS Resource Groups Tagging API	Resource Groups Tagging API
AWS RoboMaker	RoboMaker
Amazon Route 53 Domains	Route 53 Domains
Amazon Route 53 Resolver	Route 53 Resolver
Amazon S3	S3
Amazon S3 Glacier	Amazon S3 Glacier
Amazon SageMaker	Sagemaker
Savings Plans	Savings Plans
AWS Secrets Manager	Secrets Manager
AWS Security Hub	Security Hub
AWS Server Migration Service	AWS Server Migration Service
AWS Service Catalog AppRegistry	Service Catalog AppRegistry
Service Quotas	Service Quotas
AWS Shield	Shield
AWS Signer	Signer
Amazon Simple Notification Service	SNS
Amazon Simple Email Service	SES

Service	Value for the Service dimension
Amazon Simple Queue Service	SQS
AWS SSO Identity Store	Identity Store
Storage Gateway	Storage Gateway
AWS Support	Support
Amazon Simple Workflow Service	SWF
Amazon Textract	Textract
AWS IoT Things Graph	ThingsGraph
Amazon Timestream	Timestream
Amazon Transcribe	Transcribe
Amazon Translate	Translate
Amazon Transcribe streaming transcription	Transcribe Streaming
AWS Transfer Family	Transfer
AWS WAF	WAF
Amazon WorkDocs	Amazon WorkDocs
Amazon WorkLink	WorkLink
Amazon WorkMail	Amazon WorkMail
Amazon WorkSpaces	Workspaces
AWS X-Ray	X-Ray

Some services report usage metrics for additional APIs as well. To see whether an API reports usage metrics to CloudWatch, use the CloudWatch console to see the metrics reported by that service in the AWS/Usage namespace.

To see the list of a service's APIs that report usage metrics to CloudWatch

1. 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 **Usage**, and then choose **By AWS Resource**.
4. In the search box near the list of metrics, enter the name of the service. The metrics are filtered by the service you entered.

CloudWatch usage metrics

CloudWatch collects metrics that track the usage of some AWS resources. These metrics correspond to AWS service quotas. Tracking these metrics can help you proactively manage your quotas. For more information, see [Visualizing your service quotas and setting alarms \(p. 832\)](#).

Service quota usage metrics are in the AWS/Usage namespace and are collected every minute.

Currently, the only metric name in this namespace that CloudWatch publishes is `CallCount`. This metric is published with the dimensions `Resource`, `Service`, and `Type`. The `Resource` dimension specifies the name of the API operation being tracked. For example, the `CallCount` metric with the dimensions `"Service": "CloudWatch", "Type": "API"` and `"Resource": "PutMetricData"` indicates the number of times the CloudWatch `PutMetricData` API operation has been called in your account.

The `CallCount` metric does not have a specified unit. The most useful statistic for the metric is `SUM`, which represents the total operation count for the 1-minute period.

Metrics

Metric	Description
<code>CallCount</code>	The number of specified operations performed in your account.

Dimensions

Dimension	Description
<code>Service</code>	The name of the AWS service containing the resource. For CloudWatch usage metrics, the value for this dimension is <code>CloudWatch</code> .
<code>Class</code>	The class of resource being tracked. CloudWatch API usage metrics use this dimension with a value of <code>None</code> .
<code>Type</code>	The type of resource being tracked. Currently, when the <code>Service</code> dimension is <code>CloudWatch</code> , the only valid value for <code>Type</code> is <code>API</code> .
<code>Resource</code>	The name of the API operation. Valid values include the following: <code>DeleteAlarms</code> , <code>DeleteDashboards</code> , <code>DescribeAlarmHistory</code> , <code>DescribeAlarms</code> , <code>GetDashboard</code> , <code>GetMetricData</code> , <code>GetMetricStatistics</code> , <code>ListMetrics</code> , <code>PutDashboard</code> , and <code>PutMetricData</code>

CloudWatch tutorials

The following scenarios illustrate uses of Amazon CloudWatch. In the first scenario, you use the CloudWatch console to create a billing alarm that tracks your AWS usage and lets you know when you have exceeded a certain spending threshold. In the second, more advanced scenario, you use the AWS Command Line Interface (AWS CLI) to publish a single metric for a hypothetical application named *GetStarted*.

Scenarios

- [Monitor your estimated charges \(p. 840\)](#)
- [Publish metrics \(p. 842\)](#)

Scenario: Monitor your estimated charges using CloudWatch

In this scenario, you create an Amazon CloudWatch alarm to monitor your estimated charges. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

Billing metric data is stored in the US East (N. Virginia) Region and reflects worldwide charges. This data includes the estimated charges for every service in AWS that you use, as well as the estimated overall total of your AWS charges.

You can choose to receive alerts by email when charges have exceeded a certain threshold. These alerts are triggered by CloudWatch and messages are sent using Amazon Simple Notification Service (Amazon SNS).

Tasks

- [Step 1: Enable billing alerts \(p. 840\)](#)
- [Step 2: Create a billing alarm \(p. 841\)](#)
- [Step 3: Check the alarm status \(p. 842\)](#)
- [Step 4: Edit a billing alarm \(p. 842\)](#)
- [Step 5: Delete a billing alarm \(p. 842\)](#)

Step 1: Enable billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated AWS charges and create an alarm using billing metric data. After you enable billing alerts, you cannot disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using account root user credentials or as an IAM user that has been given permission to view billing information.

- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see [AWS Partner Network](#).

To enable monitoring of your estimated charges

1. Open the AWS Billing console at <https://console.aws.amazon.com/billing/>.
2. In the navigation pane, choose **Preferences**.
3. Select **Receive Billing Alerts**.
4. Choose **Save preferences**.

Step 2: Create a billing alarm

Important

Before you can create a billing alarm, you must enable billing alerts in your account, or in the management/payer account if you are using consolidated billing. For more information, see [Enabling billing alerts \(p. 159\)](#).

After you've enabled billing alerts, you can create a billing alarm. In this scenario, you create an alarm that sends an email message when your estimated charges for AWS exceed a specified threshold.

Note

This procedure uses the simple options. To use the advanced options, see [Creating a billing alarm \(p. 160\)](#) in *Create a Billing Alarm to Monitor Your Estimated AWS Charges*.

To create a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms, Create Alarm**.
4. Choose **Select metric, Billing, Total Estimated Charge**.

If you don't see **Billing** or the **Total Estimated Charge** metric, you might need to enable billing alerts. For more information, see [Step 1: Enable billing alerts \(p. 840\)](#).

5. Select the checkbox next to **EstimatedCharges** and choose **Select metric**
6. For **Whenever my total AWS charges for the month exceed**, specify the monetary amount (for example, 200) that must be exceeded to trigger the alarm and send an email notification. Then choose **Next**.

Tip

The graph shows a current estimate of your charges that you can use to set an appropriate amount.

7. For **send a notification to**, do one of the following:
 - Choose **Select an existing SNS topic** and then select the topic to notify under **Send a notification to**.
 - Choose **Create a new topic** and then type a name for the new SNS topic and enter the email addresses that are to receive the notifications. Separate the email names with commas.

8. Choose **Create Alarm**.

Step 3: Check the alarm status

Now, check the status of the billing alarm that you just created.

To check the alarm status

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**.
4. Select the check box next to the alarm. Until the subscription is confirmed, it is shown as "Pending confirmation". After the subscription is confirmed, refresh the console to show the updated status.

Step 4: Edit a billing alarm

For example, you may want to increase the amount of money you spend with AWS each month from \$200 to \$400. You can edit your existing billing alarm and increase the monetary amount that must be exceeded before the alarm is triggered.

To edit a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**.
4. Select the check box next to the alarm and choose **Actions, Modify**.
5. For **Whenever my total AWS charges for the month exceed**, specify the new amount that must be exceeded to trigger the alarm and send an email notification.
6. Choose **Save Changes**.

Step 5: Delete a billing alarm

If you no longer need your billing alarm, you can delete it.

To delete a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**.
4. Select the check box next to the alarm and choose **Actions, Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Scenario: Publish metrics to CloudWatch

In this scenario, you use the AWS Command Line Interface (AWS CLI) to publish a single metric for a hypothetical application named *GetStarted*. If you haven't already installed and configured the AWS

CLI, see [Getting Set Up with the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

Tasks

- [Step 1: Define the data configuration \(p. 843\)](#)
- [Step 2: Add metrics to CloudWatch \(p. 843\)](#)
- [Step 3: Get statistics from CloudWatch \(p. 844\)](#)
- [Step 4: View graphs with the console \(p. 844\)](#)

Step 1: Define the data configuration

In this scenario, you publish data points that track the request latency for the application. Choose names for your metric and namespace that make sense to you. For this example, name the metric *RequestLatency* and place all of the data points into the *GetStarted* namespace.

You publish several data points that collectively represent three hours of latency data. The raw data comprises 15 request latency readings distributed over three hours. Each reading is in milliseconds:

- Hour one: 87, 51, 125, 235
- Hour two: 121, 113, 189, 65, 89
- Hour three: 100, 47, 133, 98, 100, 328

You can publish data to CloudWatch as single data points or as an aggregated set of data points called a *statistic set*. You can aggregate metrics to a granularity as low as one minute. You can publish the aggregated data points to CloudWatch as a set of statistics with four predefined keys: **Sum**, **Minimum**, **Maximum**, and **SampleCount**.

You publish the data points from hour one as single data points. For the data from hours two and three, you aggregate the data points and publish a statistic set for each hour. The key values are shown in the following table.

Hour	Raw data	Sum	Minimum	Maximum	SampleCount
1	87				
1	51				
1	125				
1	235				
2	121, 113, 189, 65, 89	577	65	189	5
3	100, 47, 133, 98, 100, 328	806	47	328	6

Step 2: Add metrics to CloudWatch

After you have defined your data configuration, you are ready to add data.

To publish data points to CloudWatch

1. At a command prompt, run the following [put-metric-data](#) commands to add data for the first hour. Replace the example timestamp with a timestamp that is two hours in the past, in Universal Coordinated Time (UTC).

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 87 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 51 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 125 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 235 --unit Milliseconds
```

2. Add data for the second hour, using a timestamp that is one hour later than the first hour.

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T21:30:00Z --statistic-values
Sum=577,Minimum=65,Maximum=189,SampleCount=5 --unit Milliseconds
```

3. Add data for the third hour, omitting the timestamp to default to the current time.

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--statistic-values Sum=806,Minimum=47,Maximum=328,SampleCount=6 --unit Milliseconds
```

Step 3: Get statistics from CloudWatch

Now that you have published metrics to CloudWatch, you can retrieve statistics based on those metrics using the [get-metric-statistics](#) command as follows. Be sure to specify `--start-time` and `--end-time` far enough in the past to cover the earliest timestamp that you published.

```
aws cloudwatch get-metric-statistics --namespace GetStarted --metric-name RequestLatency \
--statistics Average \
--start-time 2016-10-14T00:00:00Z --end-time 2016-10-15T00:00:00Z --period 60
```

The following is example output:

```
{
  "Datapoints": [],
  "Label": "Request:Latency"
}
```

Step 4: View graphs with the console

After you have published metrics to CloudWatch, you can use the CloudWatch console to view statistical graphs.

To view graphs of your statistics on the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the **Navigation** pane, choose **Metrics**.
3. On the **All metrics** tab, in the search box, type **RequestLatency** and press Enter.
4. Select the check box for the **RequestLatency** metric. A graph of the metric data is displayed in the upper pane.

For more information, see [Graphing metrics \(p. 85\)](#).

Tagging your Amazon CloudWatch resources

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

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following:

- 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 CloudWatch rule that you assign to an EC2 instance.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see [Use Cost Allocation Tags](#) in the [AWS Billing User Guide](#).

The following sections provide more information about tags for CloudWatch.

Supported resources in CloudWatch

The following resources in CloudWatch support tagging:

- Alarms – You can tag alarms using the [tag-resource](#) AWS CLI command and the [TagResource](#) API.
- Canaries – You can tag canaries using the CloudWatch console. For more information, see [Creating a canary \(p. 173\)](#).
- Contributor Insights rules – You can tag Contributor Insights rules when you create them by using the [put-insight-rule](#) AWS CLI command and the [PutInsightRule](#) API. You can add tags to existing rules by using the [tag-resource](#) AWS CLI command and the [TagResource](#) API.
- Metric streams – You can tag metric streams when you create them by using the [put-metric-stream](#) AWS CLI command and the [PutMetricStream](#) API. You can add tags to existing metric streams by using the [tag-resource](#) AWS CLI command and the [TagResource](#) API.

For information about adding and managing tags, see [Managing tags \(p. 845\)](#).

Managing tags

Tags consist of the `Key` and `Value` properties on a resource. You can use the CloudWatch console, the AWS CLI, or the CloudWatch API to add, edit, or delete the values for these properties. For information about working with tags, see the following:

- [TagResource](#), [UntagResource](#), and [ListTagsForResource](#) in the [Amazon CloudWatch API Reference](#)
- [tag-resource](#), [untag-resource](#), and [list-tags-for-resource](#) in the [Amazon CloudWatch CLI Reference](#)

- [Working with Tag Editor](#) in the *Resource Groups User Guide*

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with CloudWatch resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: . : + = @ _ / - (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter` and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws:` prefix is prohibited for tags because it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags per resource limit.

Code examples for CloudWatch using AWS SDKs

The following code examples show how to use CloudWatch with an AWS software development kit (SDK).

The examples are divided into the following categories:

Actions

Code excerpts that show you how to call individual service functions.

Scenarios

Code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Actions for CloudWatch using AWS SDKs \(p. 847\)](#)
 - [Create a CloudWatch alarm that watches a metric using an AWS SDK \(p. 848\)](#)
 - [Delete CloudWatch alarms using an AWS SDK \(p. 853\)](#)
 - [Describe a CloudWatch alarm's history using an AWS SDK \(p. 857\)](#)
 - [Describe CloudWatch alarms for a metric using an AWS SDK \(p. 859\)](#)
 - [Disable CloudWatch alarm actions using an AWS SDK \(p. 863\)](#)
 - [Enable CloudWatch alarm actions using an AWS SDK \(p. 867\)](#)
 - [Get the details of a CloudWatch dashboard \(p. 873\)](#)
 - [Get CloudWatch metric statistics using an AWS SDK \(p. 874\)](#)
 - [List CloudWatch dashboards \(p. 875\)](#)
 - [List CloudWatch metrics using an AWS SDK \(p. 877\)](#)
 - [Put a set of data into a CloudWatch metric using an AWS SDK \(p. 882\)](#)
 - [Put data into a CloudWatch metric using an AWS SDK \(p. 883\)](#)
- [Scenarios for CloudWatch using AWS SDKs \(p. 887\)](#)
 - [Manage CloudWatch metrics and alarms using an AWS SDK \(p. 887\)](#)

Actions for CloudWatch using AWS SDKs

The following code examples demonstrate how to perform individual CloudWatch actions with AWS SDKs. These excerpts call the CloudWatch API and are not intended to be run in isolation. Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

The following examples include only the most commonly used actions. For a complete list, see the [CloudWatch API Reference](#).

Examples

- [Create a CloudWatch alarm that watches a metric using an AWS SDK \(p. 848\)](#)

- [Delete CloudWatch alarms using an AWS SDK \(p. 853\)](#)
- [Describe a CloudWatch alarm's history using an AWS SDK \(p. 857\)](#)
- [Describe CloudWatch alarms for a metric using an AWS SDK \(p. 859\)](#)
- [Disable CloudWatch alarm actions using an AWS SDK \(p. 863\)](#)
- [Enable CloudWatch alarm actions using an AWS SDK \(p. 867\)](#)
- [Get the details of a CloudWatch dashboard \(p. 873\)](#)
- [Get CloudWatch metric statistics using an AWS SDK \(p. 874\)](#)
- [List CloudWatch dashboards \(p. 875\)](#)
- [List CloudWatch metrics using an AWS SDK \(p. 877\)](#)
- [Put a set of data into a CloudWatch metric using an AWS SDK \(p. 882\)](#)
- [Put data into a CloudWatch metric using an AWS SDK \(p. 883\)](#)

Create a CloudWatch alarm that watches a metric using an AWS SDK

The following code examples show how to create an Amazon CloudWatch alarm that watches a metric.

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Create the alarm to watch the metric.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::PutMetricAlarmRequest request;
request.SetAlarmName(alarm_name);
request.SetComparisonOperator(
    Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
request.SetEvaluationPeriods(1);
request.SetMetricName("CPUUtilization");
request.SetNamespace("AWS/EC2");
request.SetPeriod(60);
request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
request.SetThreshold(70.0);
request.SetActionsEnabled(false);
request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);

Aws::CloudWatch::Model::Dimension dimension;
dimension.SetName("InstanceId");
dimension.SetValue(instanceId);

request.AddDimensions(dimension);

auto outcome = cw.PutMetricAlarm(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch alarm:" <<
```

```
        outcome.GetError().GetMessage() << std::endl;
    }
    else
    {
        std::cout << "Successfully created CloudWatch alarm " << alarm_name
              << std::endl;
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricAlarm](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
public static void putMetricAlarm(CloudWatchClient cw, String alarmName, String
instanceId) {

    try {
        Dimension dimension = Dimension.builder()
            .name("InstanceId")
            .value(instanceId).build();

        PutMetricAlarmRequest request = PutMetricAlarmRequest.builder()
            .alarmName(alarmName)
            .comparisonOperator(
                ComparisonOperator.GREATER_THAN_THRESHOLD)
            .evaluationPeriods(1)
            .metricName("CPUUtilization")
            .namespace("AWS/EC2")
            .period(60)
            .statistic(Statistic.AVERAGE)
            .threshold(70.0)
            .actionsEnabled(false)
            .alarmDescription(
                "Alarm when server CPU utilization exceeds 70%")
            .unit(StandardUnit.SECONDS)
            .dimensions(dimension)
            .build();

        cw.putMetricAlarm(request);
        System.out.printf(
            "Successfully created alarm with name %s", alarmName);

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricAlarm](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch service client object.
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { cwClient } from "./libs/cloudWatchClient.js";

// Set the parameters
export const params = {
    AlarmName: "Web_Server_CPU_Utilization",
    ComparisonOperator: "GreaterThanOrEqualToThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
        {
            Name: "InstanceId",
            Value: "INSTANCE_ID",
        },
    ],
    Unit: "Percent",
};

export const run = async () => {
    try {
        const data = await cwClient.send(new PutMetricAlarmCommand(params));
        console.log("Success", data);
        return data;
    } catch (err) {
        console.log("Error", err);
    }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricAlarm](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatch service object
```

```
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});

var params = {
  AlarmName: 'Web_Server_CPU_Utilization',
  ComparisonOperator: 'GreaterThanOrEqualToThreshold',
  EvaluationPeriods: 1,
  MetricName: 'CPUUtilization',
  Namespace: 'AWS/EC2',
  Period: 60,
  Statistic: 'Average',
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: 'Alarm when server CPU exceeds 70%',
  Dimensions: [
    {
      Name: 'InstanceId',
      Value: 'INSTANCE_ID'
    },
  ],
  Unit: 'Percent'
};

cw.putMetricAlarm(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricAlarm](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun putMetricAlarm(alarmNameVal: String, instanceIdVal: String) {

    val dimensionOb = Dimension {
        name = "InstanceId"
        value = instanceIdVal
    }

    val request = PutMetricAlarmRequest {
        alarmName = alarmNameVal
        comparisonOperator = ComparisonOperator.GreaterThanOrEqualToThreshold
        evaluationPeriods = 1
        metricName = "CPUUtilization"
        namespace = "AWS/EC2"
        period = 60
        statistic = Statistic.fromValue("Average")
        threshold = 70.0
        actionsEnabled = false
        alarmDescription = "An Alarm created by the Kotlin SDK when server CPU
        utilization exceeds 70%"
    }
}
```

```
        unit  = StandardUnit.fromValue("Seconds")
        dimensions = listOf(dimensionOb)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println( "Successfully created an alarm with name $alarmNameVal")
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricAlarm](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def create_metric_alarm(
            self, metric_namespace, metric_name, alarm_name, stat_type, period,
            eval_periods, threshold, comparison_op):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over the
                           alarm threshold before the alarm is set into an
                           alarmed
                           state.
        :param threshold: The threshold value to compare against the metric
                          statistic.
        :param comparison_op: The comparison operation used to compare the
                              threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op)
            logger.info(
                "Added alarm %s to track metric %s.%s.", alarm_name,
                metric_namespace,
                metric_name)
        except Exception as e:
            logger.error("Failed to create alarm %s: %s", alarm_name, str(e))
            raise e
```

```
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s", alarm_name,
metric_namespace,
                metric_name)
            raise
        else:
            return alarm
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricAlarm](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete CloudWatch alarms using an AWS SDK

The following code examples show how to delete Amazon CloudWatch alarms.

.NET

AWS SDK for .NET

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// This example shows how to delete Amazon CloudWatch alarms. The example
/// was created using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class DeleteAlarms
{
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();

        var alarmNames = CreateAlarmNameList();
        await DeleteAlarmsAsyncExample(cwClient, alarmNames);
    }

    /// <summary>
    /// Delete the alarms whose names are listed in the alarmNames parameter.
    /// </summary>
    /// <param name="client">The initialized Amazon CloudWatch client.</param>
    /// <param name="alarmNames">A list of names for the alarms to be
    /// deleted.</param>
    public static async Task DeleteAlarmsAsyncExample(IAmazonCloudWatch client,
List<string> alarmNames)
    {
        var request = new DeleteAlarmsRequest
        {
            AlarmNames = alarmNames,
        };

        try
```

```
{  
    var response = await client.DeleteAlarmsAsync(request);  
  
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)  
    {  
        Console.WriteLine("Alarms successfully deleted:");  
        alarmNames  
            .ForEach(name => Console.WriteLine($"{name}"));  
    }  
}  
catch (ResourceNotFoundException ex)  
{  
    Console.WriteLine($"Error: {ex.Message}");  
}  
}  
  
/// <summary>  
/// Defines and returns the list of alarm names to delete.  
/// </summary>  
/// <returns>A list of alarm names.</returns>  
public static List<string> CreateAlarmNameList()  
{  
    // The list of alarm names passed to DeleteAlarmsAsync  
    // can contain up to 100 alarm names.  
    var theList = new List<string>  
    {  
        "ALARM_NAME_1",  
        "ALARM_NAME_2",  
    };  
  
    return theList;  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteAlarms](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>  
#include <aws/monitoring/CloudWatchClient.h>  
#include <aws/monitoring/model/DeleteAlarmsRequest.h>  
#include <iostream>
```

Delete the alarm.

```
Aws::CloudWatch::CloudWatchClient cw;  
Aws::CloudWatch::Model::DeleteAlarmsRequest request;  
request.AddAlarmNames(alarm_name);  
  
auto outcome = cw.DeleteAlarms(request);  
if (!outcome.IsSuccess())  
{  
    std::cout << "Failed to delete CloudWatch alarm:" <<  
        outcome.GetError().GetMessage() << std::endl;
```

```
        }
    else
    {
        std::cout << "Successfully deleted CloudWatch alarm " << alarm_name
        << std::endl;
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteAlarms](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
public static void deleteCWAAlarm(CloudWatchClient cw, String alarmName) {

    try {
        DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
            .alarmNames(alarmName)
            .build();

        cw.deleteAlarms(request);
        System.out.printf("Successfully deleted alarm %s", alarmName);

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteAlarms](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch service client object.
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { cwClient } from "./libs/cloudWatchClient.js";

// Set the parameters
```

```
export const params = { AlarmNames: "ALARM_NAME" }; // e.g.,  
"Web_Server_CPU_Utilization"

export const run = async () => {
  try {
    const data = await cwClient.send(new DeleteAlarmsCommand(params));
    console.log("Success, alarm deleted; requestID:", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};

// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAlarms](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript V2

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatch service object
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});

var params = {
  AlarmNames: ['Web_Server_CPU_Utilization']
};

cw.deleteAlarms(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAlarms](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun deleteCWAAlarm(alarmNameVal: String) {
```

```
    val request = DeleteAlarmsRequest {
        alarmNames = listOf(alarmNameVal)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteAlarms](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name)
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.", metric_namespace,
                metric_name)
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DeleteAlarms](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Describe a CloudWatch alarm's history using an AWS SDK

The following code example shows how to describe Amazon CloudWatch alarm history.

.NET

AWS SDK for .NET

Get a list of CloudWatch alarms, then retrieve the history for each alarm.

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// This example retrieves a list of Amazon CloudWatch alarms and, for
/// each one, displays its history. The example was created using the
/// AWS SDK for .NET 3.7 and .NET Core 5.0.
/// </summary>
public class DescribeAlarmHistories
{
    /// <summary>
    /// Retrieves a list of alarms and then passes each name to the
    /// DescribeAlarmHistoriesAsync method to retrieve its history.
    /// </summary>
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();
        var response = await cwClient.DescribeAlarmsAsync();

        foreach (var alarm in response.MetricAlarms)
        {
            await DescribeAlarmHistoriesAsync(cwClient, alarm.AlarmName);
        }
    }

    /// <summary>
    /// Retrieves the CloudWatch alarm history for the alarm name passed
    /// to the method.
    /// </summary>
    /// <param name="client">An initialized CloudWatch client object.</param>
    /// <param name="alarmName">The CloudWatch alarm for which to retrieve
    /// history information.</param>
    public static async Task DescribeAlarmHistoriesAsync(IAmazonCloudWatch
client, string alarmName)
    {
        var request = new DescribeAlarmHistoryRequest
        {
            AlarmName = alarmName,
            EndDateUtc = DateTime.Today,
            HistoryItemType = HistoryItemType.Action,
            MaxRecords = 1,
            StartDateUtc = DateTime.Today.Subtract(TimeSpan.FromDays(30)),
        };

        var response = new DescribeAlarmHistoryResponse();

        do
        {
            response = await client.DescribeAlarmHistoryAsync(request);

            foreach (var item in response.AlarmHistoryItems)
            {
                Console.WriteLine(item.AlarmName);
                Console.WriteLine(item.HistorySummary);
                Console.WriteLine();
            }
        }
    }
}
```

```
        request.NextToken = response.NextToken;
    }
    while (!string.IsNullOrEmpty(response.NextToken));
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DescribeAlarmHistory](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Describe CloudWatch alarms for a metric using an AWS SDK

The following code examples show how to describe Amazon CloudWatch alarms for a metric.

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DescribeAlarmsRequest.h>
#include <aws/monitoring/model/DescribeAlarmsResult.h>
#include <iomanip>
#include <iostream>
```

Describe the alarms.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::DescribeAlarmsRequest request;
request.SetMaxRecords(1);

bool done = false;
bool header = false;
while (!done)
{
    auto outcome = cw.DescribeAlarms(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to describe CloudWatch alarms:" <<
            outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header)
    {
        std::cout << std::left <<
            std::setw(32) << "Name" <<
            std::setw(64) << "Arn" <<
            std::setw(64) << "Description" <<
```

```
        std::setw(20) << "LastUpdated" <<
        std::endl;
    header = true;
}

const auto &alarms = outcome.GetResult().GetMetricAlarms();
for (const auto &alarm : alarms)
{
    std::cout << std::left <<
        std::setw(32) << alarm.GetAlarmName() <<
        std::setw(64) << alarm.GetAlarmArn() <<
        std::setw(64) << alarm.GetAlarmDescription() <<
        std::setw(20) <<
        alarm.GetAlarmConfigurationUpdatedTimestamp().ToGmtString(
            SIMPLE_DATE_FORMAT_STR) <<
        std::endl;
}

const auto &next_token = outcome.GetResult().GetNextToken();
request.SetNextToken(next_token);
done = next_token.empty();
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
public static void desCWAlarms( CloudWatchClient cw) {

    try {

        boolean done = false;
        String newToken = null;

        while(!done) {
            DescribeAlarmsResponse response;

            if (newToken == null) {
                DescribeAlarmsRequest request =
DescribeAlarmsRequest.builder().build();
                response = cw.describeAlarms(request);
            } else {
                DescribeAlarmsRequest request = DescribeAlarmsRequest.builder()
                    .nextToken(newToken)
                    .build();
                response = cw.describeAlarms(request);
            }

            for(MetricAlarm alarm : response.metricAlarms()) {
                System.out.printf("\n Retrieved alarm %s", alarm.alarmName());
            }

            if(response.nextToken() == null) {
                done = true;
            } else {
                newToken = response.nextToken();
            }
        }
    }
}
```

```
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    System.out.printf("Done");
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch service client object.
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { cwClient } from "./libs/cloudWatchClient.js";

// Set the parameters
export const params = { StateValue: "INSUFFICIENT_DATA" };

export const run = async () => {
    try {
        const data = await cwClient.send(new DescribeAlarmsCommand(params));
        console.log("Success", data);
        return data;
    } catch (err) {
        console.log("Error", err);
    }
};

// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatch service object
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});

cw.describeAlarms({StateValue: 'INSUFFICIENT_DATA'}, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DescribeAlarmsForMetric](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun desCWAAlarms() {

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(DescribeAlarmsRequest {})
        response.metricAlarms?.forEach { alarm -
            println("Retrieved alarm ${alarm.alarmName}")
        }
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DescribeAlarmsForMetric](#) in [AWS SDK for Kotlin API reference](#).

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource
```

```
def get_metric_alarms(self, metric_namespace, metric_name):
    """
    Gets the alarms that are currently watching the specified metric.

    :param metric_namespace: The namespace of the metric.
    :param metric_name: The name of the metric.
    :returns: An iterator that yields the alarms.
    """
    metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
    alarm_iter = metric.alarms.all()
    logger.info("Got alarms for metric %s.%s.", metric_namespace, metric_name)
    return alarm_iter
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Disable CloudWatch alarm actions using an AWS SDK

The following code examples show how to disable Amazon CloudWatch alarm actions.

.NET

AWS SDK for .NET

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// This example shows how to disable the Amazon CloudWatch actions for
/// one or more CloudWatch alarms. The example was created using the
/// AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class DisableAlarmActions
{
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();
        var alarmNames = new List<string>
        {
            "ALARM_NAME",
            "ALARM_NAME_2",
        };

        var success = await DisableAlarmsActionsAsync(cwClient, alarmNames);

        if (success)
        {
            Console.WriteLine("Alarm action(s) successfully disabled.");
        }
        else
        {
```

```
        Console.WriteLine("Alarm action(s) were not disabled.")
    }
}

/// <summary>
/// Disable the actions for the list of CloudWatch alarm names passed
/// in the alarmNames parameter.
/// </summary>
/// <param name="client">An initialized CloudWatch client object.</param>
/// <param name="alarmNames">The list of CloudWatch alarms to disable.</param>
public static async Task<bool> DisableAlarmsActionsAsync(
    IAmazonCloudWatch client,
    List<string> alarmNames)
{
    var request = new DisableAlarmActionsRequest
    {
        AlarmNames = alarmNames,
    };

    var response = await client.DisableAlarmActionsAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DisableAlarmActions](#) in [AWS SDK for .NET API Reference](#).

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DisableAlarmActionsRequest.h>
#include <iostream>
```

Disable the alarm actions.

```
Aws::CloudWatch::CloudWatchClient cw;

Aws::CloudWatch::Model::DisableAlarmActionsRequest
disableAlarmActionsRequest;
disableAlarmActionsRequest.AddAlarmNames(alarm_name);

auto disableAlarmActionsOutcome =
cw.DisableAlarmActions(disableAlarmActionsRequest);
if (!disableAlarmActionsOutcome.IsSuccess())
{
    std::cout << "Failed to disable actions for alarm " << alarm_name <<
    ":" << disableAlarmActionsOutcome.GetError().GetMessage() <<
    std::endl;
}
else
{
```

```
        std::cout << "Successfully disabled actions for alarm " <<
                     alarm_name << std::endl;
    }
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DisableAlarmActions](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
public static void disableActions(CloudWatchClient cw, String alarmName) {

    try {
        DisableAlarmActionsRequest request =
        DisableAlarmActionsRequest.builder()
            .alarmNames(alarmName)
            .build();

        cw.disableAlarmActions(request);
        System.out.printf(
            "Successfully disabled actions on alarm %s", alarmName);

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DisableAlarmActions](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch service client object.
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { cwClient } from "./libs/cloudWatchClient.js";

// Set the parameters
```

```
export const params = { AlarmNames: "ALARM_NAME" }; // e.g.,  
"Web_Server_CPU_Utilization"  
  
export const run = async () => {  
    try {  
        const data = await cwClient.send(new DisableAlarmActionsCommand(params));  
        console.log("Success, alarm disabled:", data);  
        return data;  
    } catch (err) {  
        console.log("Error", err);  
    }  
};  
// Uncomment this line to run execution within this file.  
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DisableAlarmActions](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript V2

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js  
var AWS = require('aws-sdk');  
// Set the region  
AWS.config.update({region: 'REGION'});  
  
// Create CloudWatch service object  
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});  
  
cw.disableAlarmActions({AlarmNames: ['Web_Server_CPU_Utilization']}, function(err,  
    data) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data);  
    }  
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DisableAlarmActions](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun disableActions(alarmName: String) {  
  
    val request = DisableAlarmActionsRequest {  
        alarmNames = listOf(alarmName)  
    }  
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
```

```
        cwClient.disableAlarmActions(request)
        println("Successfully disabled actions on alarm ${alarmName}")
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DisableAlarmActions](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
        they
                           disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.", "Enabled" if enable else "Disabled",
                alarm_name)
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.", "enable" if enable else "disable",
                alarm_name)
            raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [DisableAlarmActions](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Enable CloudWatch alarm actions using an AWS SDK

The following code examples show how to enable Amazon CloudWatch alarm actions.

.NET

AWS SDK for .NET

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// This example shows how to enable the Amazon CloudWatch actions for
/// one or more CloudWatch alarms. The example was created using the
/// AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class EnableAlarmActions
{
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();
        var alarmNames = new List<string>
        {
            "ALARM_NAME",
            "ALARM_NAME_2",
        };

        var success = await EnableAlarmActionsAsync(cwClient, alarmNames);

        if (success)
        {
            Console.WriteLine("Alarm action(s) successfully enabled.");
        }
        else
        {
            Console.WriteLine("Alarm action(s) were not enabled.")
        }
    }

    /// <summary>
    /// Enable the actions for the list of CloudWatch alarm names passed
    /// in the alarmNames parameter.
    /// </summary>
    /// <param name="client">An initialized CloudWatch client object.</param>
    /// <param name="alarmNames">The list of CloudWatch alarms to enable.</param>
    /// <returns>A Boolean value indicating the success of the call.</returns>
    public static async Task<bool> EnableAlarmActionsAsync(IAmazonCloudWatch
client, List<string> alarmNames)
    {
        var request = new EnableAlarmActionsRequest
        {
            AlarmNames = alarmNames,
        };

        var response = await client.EnableAlarmActionsAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Find instructions and more code on [GitHub](#).

- For API details, see [EnableAlarmActions](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/EnableAlarmActionsRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Enable the alarm actions.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::PutMetricAlarmRequest request;
request.SetAlarmName(alarm_name);
request.SetComparisonOperator(
    Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
request.SetEvaluationPeriods(1);
request.SetMetricName("CPUUtilization");
request.SetNamespace("AWS/EC2");
request.SetPeriod(60);
request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
request.SetThreshold(70.0);
request.SetActionsEnabled(false);
request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);
request.AddAlarmActions(actionArn);

Aws::CloudWatch::Model::Dimension dimension;
dimension.SetName("InstanceId");
dimension.SetValue(instanceId);
request.AddDimensions(dimension);

auto outcome = cw.PutMetricAlarm(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch alarm:" <<
        outcome.GetError().GetMessage() << std::endl;
    return;
}

Aws::CloudWatch::Model::EnableAlarmActionsRequest enable_request;
enable_request.AddAlarmNames(alarm_name);

auto enable_outcome = cw.EnableAlarmActions(enable_request);
if (!enable_outcome.IsSuccess())
{
    std::cout << "Failed to enable alarm actions:" <<
        enable_outcome.GetError().GetMessage() << std::endl;
    return;
}

std::cout << "Successfully created alarm " << alarm_name <<
    " and enabled actions on it." << std::endl;
```

- Find instructions and more code on [GitHub](#).

- For API details, see [EnableAlarmActions](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
public static void enableActions(CloudWatchClient cw, String alarm) {  
  
    try {  
        EnableAlarmActionsRequest request = EnableAlarmActionsRequest.builder()  
            .alarmNames(alarm).build();  
  
        cw.enableAlarmActions(request);  
        System.out.printf(  
            "Successfully enabled actions on alarm %s", alarm);  
  
    } catch (CloudWatchException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [EnableAlarmActions](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon CloudWatch service client object.  
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js  
import {  
    PutMetricAlarmCommand,  
    EnableAlarmActionsCommand,  
} from "@aws-sdk/client-cloudwatch";  
import { cwClient } from "./libs/cloudWatchClient.js";  
  
// Set the parameters  
export const params = {  
    AlarmName: "ALARM_NAME", //ALARM_NAME  
    ComparisonOperator: "GreaterThanOrEqualToThreshold",  
    EvaluationPeriods: 1,  
    MetricName: "CPUUtilization",  
    Namespace: "AWS/EC2",  
    Period: 60,  
    Statistic: "Average",
```

```

    Threshold: 70.0,
    ActionsEnabled: true,
    AlarmActions: ["ACTION_ARN"], //e.g., "arn:aws:automate:us-east-1:ec2:stop"
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: "INSTANCE_ID",
      },
    ],
    Unit: "Percent",
  };

  export const run = async () => {
    try {
      const data = await cwClient.send(new PutMetricAlarmCommand(params));
      console.log("Alarm action added; RequestID:", data);
      return data;
    } catch (err) {
      console.log("Error", err);
    }
  };
  try {
    const data = await cwClient.send(
      new EnableAlarmActionsCommand(paramsEnableAlarmAction)
    );
    console.log("Alarm action enabled; RequestID:", data.$metadata.requestId);
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();

```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [EnableAlarmActions](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript V2

Import the SDK and client modules and call the API.

```

// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatch service object
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});

var params = {
  AlarmName: 'Web_Server_CPU_Utilization',
  ComparisonOperator: 'GreaterThanOrEqualToThreshold',
  EvaluationPeriods: 1,
  MetricName: 'CPUUtilization',
  Namespace: 'AWS/EC2',
  Period: 60,
  Statistic: 'Average',
  Threshold: 70.0,
  ActionsEnabled: true,
}

```

```
AlarmActions: ['ACTION_ARN'],
AlarmDescription: 'Alarm when server CPU exceeds 70%',
Dimensions: [
  {
    Name: 'InstanceId',
    Value: 'INSTANCE_ID'
  },
],
Unit: 'Percent'
};

cw.putMetricAlarm(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName]
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function(err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [EnableAlarmActions](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

```
suspend fun enableActions(alarm: String) {

    val request = EnableAlarmActionsRequest {
        alarmNames = listOf(alarm)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.enableAlarmActions(request)
        println( "Successfully enabled actions on alarm $alarm")
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [EnableAlarmActions](#) in [AWS SDK for Kotlin API reference](#).

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
        they
                           disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.", "Enabled" if enable else "Disabled",
                alarm_name)
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.", "enable" if enable else "disable",
                alarm_name)
        raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [EnableAlarmActions](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the details of a CloudWatch dashboard

The following code example shows how to get Amazon CloudWatch dashboard details.

.NET

AWS SDK for .NET

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
```

```
/// <summary>
/// This example shows how to retrieve the details of an Amazon CloudWatch
/// dashboard. The return value from the call to GetDashboard is a json
/// object representing the widgets in the dashboard. The example was
/// created using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class GetDashboard
{
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();
        string dashboardName = "CloudWatch-Default";

        var body = await GetDashboardAsync(cwClient, dashboardName);

        Console.WriteLine(body);
    }

    /// <summary>
    /// Get the json that represents the dashboard.
    /// </summary>
    /// <param name="client">An initialized CloudWatch client.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>The string containing the json value describing the
    /// contents and layout of the CloudWatch dashboard.</returns>
    public static async Task<string> GetDashboardAsync(IAmazonCloudWatch
client, string dashboardName)
    {

        var request = new GetDashboardRequest
        {
            DashboardName = dashboardName,
        };

        var response = await client.GetDashboardAsync(request);

        return response.DashboardBody;
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetDashboard](#) in [AWS SDK for .NET API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get CloudWatch metric statistics using an AWS SDK

The following code example shows how to get Amazon CloudWatch metric statistics.

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
```

```
def __init__(self, cloudwatch_resource):
    """
    :param cloudwatch_resource: A Boto3 CloudWatch resource.
    """
    self.cloudwatch_resource = cloudwatch_resource

    def get_metric_statistics(self, namespace, name, start, end, period,
                           stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
        grouped
            into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
        period
            must
                be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
        value
            or maximum value.
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period, Statistics=stat_types)
            logger.info(
                "Got %s statistics for %s.", len(stats['Datapoints']),
            stats['Label'])
            except ClientError:
                logger.exception("Couldn't get statistics for %s.%s.", namespace, name)
                raise
            else:
                return stats
        
```

- Find instructions and more code on [GitHub](#).
- For API details, see [GetMetricStatistics](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

List CloudWatch dashboards

The following code example shows how to list Amazon CloudWatch dashboards.

.NET

AWS SDK for .NET

```
using System;
using System.Collections.Generic;
```

```
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Shows how to retrieve a list of Amazon CloudWatch dashboards. This
/// example was written using AWSSDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class ListDashboards
{
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();
        var dashboards = await ListDashboardsAsync(cwClient);

        DisplayDashboardList(dashboards);
    }

    /// <summary>
    /// Get the list of available dashboards.
    /// </summary>
    /// <param name="client">The initialized CloudWatch client used to
    /// retrieve a list of defined dashboards.</param>
    /// <returns>A list of DashboardEntry objects.</returns>
    public static async Task<List<DashboardEntry>>
ListDashboardsAsync(IAmazonCloudWatch client)
    {
        var response = await client.ListDashboardsAsync(new
ListDashboardsRequest());
        return response.DashboardEntries;
    }

    /// <summary>
    /// Displays the name of each CloudWatch Dashboard in the list passed
    /// to the method.
    /// </summary>
    /// <param name="dashboards">A list of DashboardEntry objects.</param>
    public static void DisplayDashboardList(List<DashboardEntry> dashboards)
    {
        if (dashboards.Count > 0)
        {
            Console.WriteLine("The following dashboards are defined:");
            foreach (var dashboard in dashboards)
            {
                Console.WriteLine($"Name: {dashboard.DashboardName} Last
modified: {dashboard.LastModified}");
            }
        }
        else
        {
            Console.WriteLine("No dashboards found.");
        }
    }
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListDashboards](#) in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

List CloudWatch metrics using an AWS SDK

The following code examples show how to list Amazon CloudWatch metrics.

.NET

AWS SDK for .NET

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// This example demonstrates how to list metrics for Amazon CloudWatch.
/// The example was created using the AWS SDK for .NET version 3.7 and
/// .NET Core 5.0.
/// </summary>
public class ListMetrics
{
    public static async Task Main()
    {
        IAmazonCloudWatch cwClient = new AmazonCloudWatchClient();

        var filter = new DimensionFilter
        {
            Name = "InstanceType",
            Value = "t1.micro",
        };
        string metricName = "CPUUtilization";
        string namespaceName = "AWS/EC2";

        await ListMetricsAsync(cwClient, filter, metricName, namespaceName);
    }

    /// <summary>
    /// Retrieve CloudWatch metrics using the supplied filter, metrics name,
    /// and namespace.
    /// </summary>
    /// <param name="client">An initialized CloudWatch client.</param>
    /// <param name="filter">The filter to apply in retrieving metrics.</param>
    /// <param name="metricName">The metric name for which to retrieve
    /// information.</param>
    /// <param name="nameSpaceName">The name of the namespace from which
    /// to retrieve metric information.</param>
    public static async Task ListMetricsAsync(
        IAmazonCloudWatch client,
        DimensionFilter filter,
        string metricName,
        string nameSpaceName)
    {
        var request = new ListMetricsRequest
        {
            Dimensions = new List<DimensionFilter>() { filter },
            MetricName = metricName,
            Namespace = nameSpaceName,
        };

        var response = new ListMetricsResponse();
        do
        {
            response = await client.ListMetricsAsync(request);
```

```
        if (response.Metrics.Count > 0)
        {
            foreach (var metric in response.Metrics)
            {
                Console.WriteLine(metric.MetricName +
                    " (" + metric.Namespace + ")");

                foreach (var dimension in metric.Dimensions)
                {
                    Console.WriteLine(" " + dimension.Name + ":" +
                        dimension.Value);
                }
            }
        }
        else
        {
            Console.WriteLine("No metrics found.");
        }

        request.NextToken = response.NextToken;
    }
    while (!string.IsNullOrEmpty(response.NextToken));
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListMetrics](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/ListMetricsResult.h>
#include <iomanip>
#include <iostream>
```

List the metrics.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::ListMetricsRequest request;

if (argc > 1)
{
    request.SetMetricName(argv[1]);
}

if (argc > 2)
{
    request.SetNamespace(argv[2]);
}

bool done = false;
```

```

bool header = false;
while (!done)
{
    auto outcome = cw.ListMetrics(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to list CloudWatch metrics:" <<
            outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header)
    {
        std::cout << std::left << std::setw(48) << "MetricName" <<
            std::setw(32) << "Namespace" << "DimensionNameValuePair" <<
            std::endl;
        header = true;
    }

    const auto &metrics = outcome.GetResult().GetMetrics();
    for (const auto &metric : metrics)
    {
        std::cout << std::left << std::setw(48) <<
            metric.GetMetricName() << std::setw(32) <<
            metric.GetNamespace();
        const auto &dimensions = metric.GetDimensions();
        for (auto iter = dimensions.cbegin(); iter != dimensions.cend(); ++iter)
        {
            const auto &dimkv = *iter;
            std::cout << dimkv.GetName() << " = " << dimkv.GetValue();
            if (iter + 1 != dimensions.cend())
            {
                std::cout << ", ";
            }
        }
        std::cout << std::endl;
    }

    const auto &next_token = outcome.GetResult().GetNextToken();
    request.SetNextToken(next_token);
    done = next_token.empty();
}

```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListMetrics](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```

public static void listMets( CloudWatchClient cw, String namespace) {

    boolean done = false;
    String nextToken = null;

    try {
        while(!done) {

            ListMetricsResponse response;

```

```
if (nextToken == null) {
    ListMetricsRequest request = ListMetricsRequest.builder()
        .namespace(namespace)
        .build();

    response = cw.listMetrics(request);
} else {
    ListMetricsRequest request = ListMetricsRequest.builder()
        .namespace(namespace)
        .nextToken(nextToken)
        .build();

    response = cw.listMetrics(request);
}

for (Metric metric : response.metrics()) {
    System.out.printf(
        "Retrieved metric %s", metric.metricName());
    System.out.println();
}

if(response.nextToken() == null) {
    done = true;
} else {
    nextToken = response.nextToken();
}
}

} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListMetrics](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch service client object.
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { cwClient } from "./libs/cloudWatchClient.js";

// Set the parameters
```

```
export const params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

export const run = async () => {
  try {
    const data = await cwClient.send(new ListMetricsCommand(params));
    console.log("Success. Metrics:", JSON.stringify(data.Metrics));
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListMetrics](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatch service object
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});

var params = {
  Dimensions: [
    {
      Name: 'LogGroupName', /* required */
    },
  ],
  MetricName: 'IncomingLogEvents',
  Namespace: 'AWS/Logs'
};

cw.listMetrics(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListMetrics](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def list_metrics(self, namespace, name, recent=False):
        """
        Gets the metrics within a namespace that have the specified name.
        If the metric has no dimensions, a single metric is returned.
        Otherwise, metrics for all dimensions are returned.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param recent: When True, only metrics that have been active in the last
                       three hours are returned.
        :return: An iterator that yields the retrieved metrics.
        """
        try:
            kwargs = {'Namespace': namespace, 'MetricName': name}
            if recent:
                kwargs['RecentlyActive'] = 'PT3H' # List past 3 hours only
            metric_iter = self.cloudwatch_resource.metrics.filter(**kwargs)
            logger.info("Got metrics for %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't get metrics for %s.%s.", namespace, name)
            raise
        else:
            return metric_iter
```

- Find instructions and more code on [GitHub](#).
- For API details, see [ListMetrics](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Put a set of data into a CloudWatch metric using an AWS SDK

The following code example shows how to put a set of data into a Amazon CloudWatch metric.

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
```

```
"""
    self.cloudwatch_resource = cloudwatch_resource

def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
    """
        Sends a set of data to CloudWatch for a metric. All of the data in the set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
            contains a list of values and a list of corresponding
            counts.
            The value and count lists must be the same length.
    """
    try:
        metric = self.cloudwatch_resource.Metric(namespace, name)
        metric.put_data(
            Namespace=namespace,
            MetricData=[{
                'MetricName': name,
                'Timestamp': timestamp,
                'Values': data_set['values'],
                'Counts': data_set['counts'],
                'Unit': unit}])
        logger.info("Put data set for metric %s.%s.", namespace, name)
    except ClientError:
        logger.exception("Couldn't put data set for metric %s.%s.", namespace,
name)
        raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricData](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Put data into a CloudWatch metric using an AWS SDK

The following code examples show how to put data into a Amazon CloudWatch metric.

C++

SDK for C++

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricDataRequest.h>
#include <iostream>
```

Put data into the metric.

```
Aws::CloudWatch::CloudWatchClient cw;
```

```
Aws::CloudWatch::Model::Dimension dimension;
dimension.SetName("UNIQUE_PAGES");
dimension.SetValue("URLS");

Aws::CloudWatch::Model::MetricDatum datum;
datum.SetMetricName("PAGES_VISITED");
datum.SetUnit(Aws::CloudWatch::Model::StandardUnit::None);
datum.SetValue(data_point);
datum.AddDimensions(dimension);

Aws::CloudWatch::Model::PutMetricDataRequest request;
request.SetNamespace("SITE/TRAFFIC");
request.AddMetricData(datum);

auto outcome = cw.PutMetricData(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to put sample metric data:" <<
        outcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully put sample metric data" << std::endl;
}
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricData](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

```
public static void putMetData(CloudWatchClient cw, Double dataPoint ) {

    try {
        Dimension dimension = Dimension.builder()
            .name("UNIQUE_PAGES")
            .value("URLS")
            .build();

        // Set an Instant object
        String time =
ZonedDateTime.now( ZoneOffset.UTC ).format( DateTimeFormatter.ISO_INSTANT );
        Instant instant = Instant.parse(time);

        MetricDatum datum = MetricDatum.builder()
            .metricName("PAGES_VISITED")
            .unit(StandardUnit.NONE)
            .value(dataPoint)
            .timestamp(instant)
            .dimensions(dimension).build();

        PutMetricDataRequest request = PutMetricDataRequest.builder()
            .namespace("SITE/TRAFFIC")
            .metricData(datum).build();

        cw.putMetricData(request);

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

```
        }
        System.out.printf("Successfully put data point %f", dataPoint);
    }
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricData](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon CloudWatch service client object.
export const cwClient = new CloudWatchClient({ region: REGION });
```

Import the SDK and client modules and call the API.

```
// Import required AWS SDK clients and commands for Node.js
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { cwClient } from "./libs/cloudWatchClient.js";

// Set the parameters
export const params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

export const run = async () => {
  try {
    const data = await cwClient.send(new PutMetricDataCommand(params));
    console.log("Success", data.$metadata.requestId);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
// Uncomment this line to run execution within this file.
// run();
```

- Find instructions and more code on [GitHub](#).

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricData](#) in [AWS SDK for JavaScript API Reference](#).

SDK for JavaScript V2

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'REGION'});

// Create CloudWatch service object
var cw = new AWS.CloudWatch({apiVersion: '2010-08-01'});

// Create parameters JSON for putMetricData
var params = {
    MetricData: [
        {
            MetricName: 'PAGES_VISITED',
            Dimensions: [
                {
                    Name: 'UNIQUE_PAGES',
                    Value: 'URLS'
                },
                {
                    Unit: 'None',
                    Value: 1.0
                }
            ],
            Namespace: 'SITE/TRAFFIC'
        };
    ],
    cw.putMetricData(params, function(err, data) {
        if (err) {
            console.log("Error", err);
        } else {
            console.log("Success", JSON.stringify(data));
        }
    });
}
```

- Find instructions and more code on [GitHub](#).
- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricData](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is given
        a timestamp of the current UTC time.
        
```

```
:param namespace: The namespace of the metric.  
:param name: The name of the metric.  
:param value: The value of the metric.  
:param unit: The unit of the metric.  
"""  
try:  
    metric = self.cloudwatch_resource.Metric(namespace, name)  
    metric.put_data(  
        Namespace=namespace,  
        MetricData=[{  
            'MetricName': name,  
            'Value': value,  
            'Unit': unit  
        }]  
    )  
    logger.info("Put data for metric %s.%s", namespace, name)  
except ClientError:  
    logger.exception("Couldn't put data for metric %s.%s", namespace, name)  
    raise
```

- Find instructions and more code on [GitHub](#).
- For API details, see [PutMetricData](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for CloudWatch using AWS SDKs

The following code examples show you how to implement common scenarios in CloudWatch with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within CloudWatch. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

- [Manage CloudWatch metrics and alarms using an AWS SDK \(p. 887\)](#)

Manage CloudWatch metrics and alarms using an AWS SDK

The following code example shows how to:

- Create an alarm that watches a metric.
- Put data into a CloudWatch metric and trigger the alarm.
- Get data from the alarm.
- Delete the alarm.

Python

[SDK for Python \(Boto3\)](#)

Create a class that wraps CloudWatch operations.

```

from datetime import datetime, timedelta
import logging
from pprint import pprint
import random
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                        contains a list of values and a list of corresponding
        counts.
                        The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{
                    'MetricName': name,
                    'Timestamp': timestamp,
                    'Values': data_set['values'],
                    'Counts': data_set['counts'],
                    'Unit': unit}])
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.", namespace,
                             name)
            raise

    def create_metric_alarm(
        self, metric_namespace, metric_name, alarm_name, stat_type, period,
        eval_periods, threshold, comparison_op):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                      statistics.
        :param eval_periods: The number of periods that the metric must be over the
                            alarm threshold before the alarm is set into an
                            alarmed
                            state.
        """

```

```

        :param threshold: The threshold value to compare against the metric
        statistic.
        :param comparison_op: The comparison operation used to compare the
        threshold
                           against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op)
            logger.info(
                "Added alarm %s to track metric %s.%s.", alarm_name,
            metric_namespace,
                metric_name)
            except ClientError:
                logger.exception(
                    "Couldn't add alarm %s to metric %s.%s", alarm_name,
            metric_namespace,
                metric_name)
                raise
            else:
                return alarm

        def put_metric_data(self, namespace, name, value, unit):
            """
            Sends a single data value to CloudWatch for a metric. This metric is given
            a timestamp of the current UTC time.

            :param namespace: The namespace of the metric.
            :param name: The name of the metric.
            :param value: The value of the metric.
            :param unit: The unit of the metric.
            """
            try:
                metric = self.cloudwatch_resource.Metric(namespace, name)
                metric.put_data(
                    Namespace=namespace,
                    MetricData=[{
                        'MetricName': name,
                        'Value': value,
                        'Unit': unit
                    }])
            )
            logger.info("Put data for metric %s.%s", namespace, name)
            except ClientError:
                logger.exception("Couldn't put data for metric %s.%s", namespace, name)
                raise

        def get_metric_statistics(self, namespace, name, start, end, period,
        stat_types):
            """
            Gets statistics for a metric within a specified time span. Metrics are
            grouped
            into the specified period.

            :param namespace: The namespace of the metric.
            :param name: The name of the metric.
            :param start: The UTC start time of the time span to retrieve.
            :param end: The UTC end time of the time span to retrieve.

```

```

:param period: The period, in seconds, in which to group metrics. The
period
    must match the granularity of the metric, which depends on
    the metric's age. For example, metrics that are older than
    three hours have a one-minute granularity, so the period
must
    be at least 60 and must be a multiple of 60.
:param stat_types: The type of statistics to retrieve, such as average
value
    or maximum value.
:return: The retrieved statistics for the metric.
"""
try:
    metric = self.cloudwatch_resource.Metric(namespace, name)
    stats = metric.get_statistics(
        StartTime=start, EndTime=end, Period=period, Statistics=stat_types)
    logger.info(
        "Got %s statistics for %s.", len(stats['Datapoints']),
        stats['Label'])
except ClientError:
    logger.exception("Couldn't get statistics for %s.%s.", namespace, name)
    raise
else:
    return stats

def get_metric_alarms(self, metric_namespace, metric_name):
    """
    Gets the alarms that are currently watching the specified metric.

    :param metric_namespace: The namespace of the metric.
    :param metric_name: The name of the metric.
    :returns: An iterator that yields the alarms.
    """
    metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
    alarm_iter = metric.alarms.all()
    logger.info("Got alarms for metric %s.%s.", metric_namespace, metric_name)
    return alarm_iter

def delete_metric_alarms(self, metric_namespace, metric_name):
    """
    Deletes all of the alarms that are currently watching the specified metric.

    :param metric_namespace: The namespace of the metric.
    :param metric_name: The name of the metric.
    """
    try:
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        metric.alarms.delete()
        logger.info(
            "Deleted alarms for metric %s.%s.", metric_namespace, metric_name)
    except ClientError:
        logger.exception(
            "Couldn't delete alarms for metric %s.%s.", metric_namespace,
            metric_name)
        raise

```

Use the wrapper class to put data in a metric, trigger an alarm that watches the metric, and get data from the alarm.

```

def usage_demo():
    print('*'*88)
    print("Welcome to the Amazon CloudWatch metrics and alarms demo!")
    print('*'*88)

```

```

logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

cw_wrapper = CloudWatchWrapper(boto3.resource('cloudwatch'))

minutes = 20
metric_namespace = 'doc-example-metric'
metric_name = 'page_views'
start = datetime.utcnow() - timedelta(minutes=minutes)
print(f"Putting data into metric {metric_namespace}.{metric_name} spanning the
"
      f"last {minutes} minutes.")
for offset in range(0, minutes):
    stamp = start + timedelta(minutes=offset)
    cw_wrapper.put_metric_data_set(
        metric_namespace, metric_name, stamp, 'Count', {
            'values': [
                random.randint(bound, bound * 2)
                for bound in range(offset + 1, offset + 11)],
            'counts': [random.randint(1, offset + 1) for _ in range(10)]
        })
)

alarm_name = 'high_page_views'
period = 60
eval_periods = 2
print(f"Creating alarm {alarm_name} for metric {metric_name}.")
alarm = cw_wrapper.create_metric_alarm(
    metric_namespace, metric_name, alarm_name, 'Maximum', period, eval_periods,
    100, 'GreaterThanThreshold')
print(f"Alarm ARN is {alarm.alarm_arn}.")
print(f"Current alarm state is: {alarm.state_value}.")

print(f"Sending data to trigger the alarm. This requires data over the
threshold "
      f"for {eval_periods} periods of {period} seconds each.")
while alarm.state_value == 'INSUFFICIENT_DATA':
    print("Sending data for the metric.")
    cw_wrapper.put_metric_data(
        metric_namespace, metric_name, random.randint(100, 200), 'Count')
    alarm.load()
    print(f"Current alarm state is: {alarm.state_value}.")
    if alarm.state_value == 'INSUFFICIENT_DATA':
        print(f"Waiting for {period} seconds...")
        time.sleep(period)
    else:
        print("Wait for a minute for eventual consistency of metric data.")
        time.sleep(period)
        if alarm.state_value == 'OK':
            alarm.load()
            print(f"Current alarm state is: {alarm.state_value}.")

print(f"Getting data for metric {metric_namespace}.{metric_name} during
timespan "
      f"of {start} to {datetime.utcnow()} (times are UTC).")
stats = cw_wrapper.get_metric_statistics(
    metric_namespace, metric_name, start, datetime.utcnow(), 60,
    ['Average', 'Minimum', 'Maximum'])
print(f"Got {len(stats['Datapoints'])} data points for metric "
      f"{metric_namespace}.{metric_name}.")
pprint(sorted(stats['Datapoints'], key=lambda x: x['Timestamp']))

print(f"Getting alarms for metric {metric_name}.")
alarms = cw_wrapper.get_metric_alarms(metric_namespace, metric_name)
for alarm in alarms:
    print(f"Alarm {alarm.name} is currently in state {alarm.state_value}.")

```

```
print(f"Deleting alarms for metric {metric_name}.")  
cw_wrapper.delete_metric_alarms(metric_namespace, metric_name)  
  
print("Thanks for watching!")  
print('*'*88)
```

- Find instructions and more code on [GitHub](#).

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK \(p. 17\)](#). This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon CloudWatch

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 WorkSpaces, 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 Amazon CloudWatch. It shows you how to configure Amazon CloudWatch to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CloudWatch resources.

Contents

- [Data protection in Amazon CloudWatch \(p. 893\)](#)
- [Identity and access management for Amazon CloudWatch \(p. 894\)](#)
- [Compliance validation for Amazon CloudWatch \(p. 951\)](#)
- [Resilience in Amazon CloudWatch \(p. 951\)](#)
- [Infrastructure security in Amazon CloudWatch \(p. 951\)](#)
- [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints \(p. 952\)](#)
- [Security considerations for Synthetics canaries \(p. 955\)](#)

Data protection in Amazon CloudWatch

The AWS [shared responsibility model](#) applies to data protection in Amazon CloudWatch. 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 information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the [AWS Security Blog](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual 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. We recommend TLS 1.2 or later.
- 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.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

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

Encryption in transit

CloudWatch uses end-to-end encryption of data in transit.

Identity and access management for Amazon CloudWatch

Access to Amazon CloudWatch requires credentials. Those credentials must have permissions to access AWS resources, such as retrieving CloudWatch metric data about your cloud resources. The following sections provide details about how you can use [AWS Identity and Access Management \(IAM\)](#) and CloudWatch to help secure your resources by controlling who can access them:

- [Authentication \(p. 894\)](#)
- [Access control \(p. 895\)](#)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *AWS account user credentials* and they provide complete access to all of your AWS resources.

Important

For security reasons, we recommend that you use the AWS account user credentials only to create an *administrator*, which is an *IAM user* with full permissions to your account. Then, you can use this administrator to create other IAM users and roles with limited permissions. For more information, see [IAM Best Practices](#) and [Creating an Admin User and Group](#) in the *IAM User Guide*.

- **IAM user** – An *IAM user* is an identity within your AWS account that has specific custom permissions (for example, permissions to view metrics in CloudWatch). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and AWS CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. CloudWatch supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the [AWS General Reference](#).

- **IAM role** – An [IAM role](#) is another IAM identity you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:
 - **Federated user access** – Instead of creating an IAM user, you can use preexisting identities from AWS Directory Service, your enterprise user directory, or a web identity provider (IdP). These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [IdP](#). For more information, see [Federated Users and Roles](#) in the [IAM User Guide](#).
 - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the [IAM User Guide](#).
 - **AWS service access** – You can use an IAM role in your account to grant an AWS service the permissions needed to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the [IAM User Guide](#).
 - **Applications running on Amazon EC2** – Instead of storing access keys within the EC2 instance for use by applications running on the instance and making API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see [Using Roles for Applications on Amazon EC2](#) in the [IAM User Guide](#).

Access control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access CloudWatch resources. For example, you must have permissions to create CloudWatch dashboard widgets, view metrics, and so on.

The following sections describe how to manage permissions for CloudWatch. We recommend that you read the overview first.

- [Overview of managing access permissions to your CloudWatch resources \(p. 896\)](#)
- [Using identity-based policies \(IAM policies\) for CloudWatch \(p. 899\)](#)

- [Amazon CloudWatch permissions reference \(p. 942\)](#)

CloudWatch dashboard permissions update

On May 1, 2018, AWS changed the permissions required to access CloudWatch dashboards. Dashboard access in the CloudWatch console now requires permissions that were introduced in 2017 to support dashboard API operations:

- `cloudwatch:GetDashboard`
- `cloudwatch>ListDashboards`
- `cloudwatch:PutDashboard`
- `cloudwatch>DeleteDashboards`

To access CloudWatch dashboards, you need one of the following:

- The **AdministratorAccess** policy.
- The **CloudWatchFullAccess** policy.
- A custom policy that includes one or more of these specific permissions:
 - `cloudwatch:GetDashboard` and `cloudwatch>ListDashboards` to be able to view dashboards
 - `cloudwatch:PutDashboard` to be able to create or modify dashboards
 - `cloudwatch>DeleteDashboards` to be able to delete dashboards

For more information for changing permissions for an IAM user using policies, see [Changing Permissions for an IAM User](#).

For more information about CloudWatch permissions, see [Amazon CloudWatch permissions reference \(p. 942\)](#).

For more information about dashboard API operations, see [PutDashboard](#) in the Amazon CloudWatch API Reference.

Overview of managing access permissions to your CloudWatch resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

Note

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

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

- [CloudWatch resources and operations \(p. 897\)](#)
- [Understanding resource ownership \(p. 897\)](#)
- [Managing access to resources \(p. 898\)](#)
- [Specifying policy elements: Actions, effects, and principals \(p. 899\)](#)

- Specifying conditions in a policy (p. 899)

CloudWatch resources and operations

You can restrict access to specific alarms and dashboards by using their Amazon Resource Names (ARNs) in your policies. For more information, see [Actions, Resources, and Condition Keys for Amazon CloudWatch](#) in the *IAM User Guide*.

You use an * (asterisk) as the resource when writing a policy to control access to CloudWatch actions. For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["cloudwatch:GetMetricData", "cloudwatch>ListMetrics"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "true"
                }
            }
        }
    ]
}
```

For more information about ARNs, see [ARNs in IAM User Guide](#). For information about CloudWatch Logs ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#) in the *Amazon Web Services General Reference*. For an example of a policy that covers CloudWatch actions, see [Using identity-based policies \(IAM policies\) for CloudWatch \(p. 899\)](#).

Action	ARN (with Region)	ARN (for use with IAM role)
Stop	arn:aws:automate:us-east-1:ec2:stop	arn:aws:swf:us-east-1: customer-account :action/actions/AWS_EC2.InstanceId.Stop/1.0
Terminate	arn:aws:automate:us-east-1:ec2:terminate	arn:aws:swf:us-east-1: customer-account :action/actions/AWS_EC2.InstanceId.Terminate/1.0
Reboot	arn:aws:automate:us-east-1:ec2:reboot	arn:aws:swf:us-east-1: customer-account :action/actions/AWS_EC2.InstanceId.Reboot/1.0
Recover	arn:aws:automate:us-east-1:ec2:recover	arn:aws:swf:us-east-1: customer-account :action/actions/AWS_EC2.InstanceId.Recover/1.0

Understanding resource ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the [principal entity](#) (that is, the

AWS account root user, an IAM user, or an IAM role) that authenticates the resource creation request. CloudWatch does not have any resources that you can own.

Managing access to resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

Note

This section discusses using IAM in the context of CloudWatch. 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 [IAM policy reference](#) in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as identity-based policies (IAM policies) and policies attached to a resource are referred to as resource-based policies. CloudWatch supports only identity-based policies.

Topics

- [Identity-based policies \(IAM policies\) \(p. 898\)](#)
- [Resource-based policies \(IAM policies\) \(p. 898\)](#)

Identity-based policies (IAM policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create an Amazon CloudWatch resource, such as metrics, 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 can 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*.

For more information about using identity-based policies with CloudWatch, see [Using identity-based policies \(IAM policies\) for CloudWatch \(p. 899\)](#). For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

Resource-based policies (IAM policies)

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an Amazon S3 bucket to manage access permissions to that bucket. CloudWatch doesn't support resource-based policies.

Specifying policy elements: Actions, effects, and principals

For each CloudWatch resource, the service defines a set of API operations. To grant permissions for these API operations, CloudWatch defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see [CloudWatch resources and operations \(p. 897\)](#) and [CloudWatch Actions](#).

The following are the basic policy elements:

- **Resource** – Use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. CloudWatch does not have any resources for you to control using policies resources, so use the wildcard character (*) in IAM policies. For more information, see [CloudWatch resources and operations \(p. 897\)](#).
- **Action** – Use action keywords to identify resource operations that you want to allow or deny. For example, the `cloudwatch:ListMetrics` permission allows the user permissions to perform the `ListMetrics` operation.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific 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, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that 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 (applies to resource-based policies only). CloudWatch doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see [AWS IAM JSON Policy Reference](#) in the *IAM User Guide*.

For a table showing all of the CloudWatch API actions and the resources that they apply to, see [Amazon CloudWatch permissions reference \(p. 942\)](#).

Specifying conditions in a policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see [Condition](#) in the *IAM User Guide*.

To express conditions, you use predefined condition keys. For a list of context keys supported by each AWS service and a list of AWS-wide policy keys, see [Actions, resources, and condition keys for AWS services](#) and [Global and IAM Condition Context Keys](#) in the *IAM User Guide*.

Using identity-based policies (IAM policies) for CloudWatch

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on CloudWatch resources.

Important

We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your CloudWatch resources. For more information, see [Access control \(p. 895\)](#).

The sections in this topic cover the following:

- [Permissions required to use the CloudWatch console \(p. 900\)](#)
- [AWS managed \(predefined\) policies for CloudWatch \(p. 903\)](#)
- [Customer managed policy examples \(p. 915\)](#)

The following shows an example of a permissions policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["cloudwatch:GetMetricData", "cloudwatch>ListMetrics"],  
            "Resource": "*",  
            "Condition": {  
                "Bool": {  
                    "aws:SecureTransport": "true"  
                }  
            }  
        }  
    ]  
}
```

This sample policy has one statement that grants permissions to a group for two CloudWatch actions (`cloudwatch:GetMetricData`, and `cloudwatch>ListMetrics`), but only if the group uses SSL with the request (`"aws:SecureTransport": "true"`). For more information about the elements within an IAM policy statement, see [Specifying policy elements: Actions, effects, and principals \(p. 899\)](#) and [IAM Policy Elements Reference](#) in [IAM User Guide](#).

Permissions required to use the CloudWatch console

For a user to work with the CloudWatch console, that user must have a minimum set of permissions that allow the user to describe other AWS resources in their account. The CloudWatch console requires permissions from the following services:

- [Amazon EC2 Auto Scaling](#)
- [CloudTrail](#)
- [CloudWatch](#)
- [CloudWatch Events](#)
- [CloudWatch Logs](#)
- [Amazon EC2](#)
- [OpenSearch Service](#)
- [IAM](#)
- [Kinesis](#)
- [Lambda](#)
- [Amazon S3](#)
- [Amazon SNS](#)
- [Amazon SQS](#)
- [Amazon SWF](#)
- X-Ray, if you are using the ServiceLens feature

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy. To ensure that those users can still use the

CloudWatch console, also attach the `CloudWatchReadOnlyAccess` managed policy to the user, as described in [AWS managed \(predefined\) policies for CloudWatch \(p. 903\)](#).

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the CloudWatch API.

The full set of permissions required to work with the CloudWatch console are listed below:

- `application-autoscaling:DescribeScalingPolicies`
- `autoscaling:DescribeAutoScalingGroups`
- `autoscaling:DescribePolicies`
- `cloudtrail:DescribeTrails`
- `cloudwatch:DeleteAlarms`
- `cloudwatch:DescribeAlarmHistory`
- `cloudwatch:DescribeAlarms`
- `cloudwatch:GetMetricData`
- `cloudwatch:GetMetricStatistics`
- `cloudwatch>ListMetrics`
- `cloudwatch:PutMetricAlarm`
- `cloudwatch:PutMetricData`
- `ec2:DescribeInstances`
- `ec2:DescribeTags`
- `ec2:DescribeVolumes`
- `es:DescribeElasticsearchDomain`
- `es>ListDomainNames`
- `events>DeleteRule`
- `events:DescribeRule`
- `events:DisableRule`
- `events:EnableRule`
- `events>ListRules`
- `events:PutRule`
- `iam:AttachRolePolicy`
- `iam>CreateRole`
- `iam:GetPolicy`
- `iam:GetPolicyVersion`
- `iam:GetRole`
- `iam>ListAttachedRolePolicies`
- `iam>ListRoles`
- `kinesis:DescribeStream`
- `kinesis>ListStreams`
- `lambda:AddPermission`
- `lambda>CreateFunction`
- `lambda:GetFunctionConfiguration`
- `lambda>ListAliases`
- `lambda>ListFunctions`
- `lambda>ListVersionsByFunction`

- lambda:RemovePermission
- logs:CancelExportTask
- logs>CreateExportTask
- logs>CreateLogGroup
- logs>CreateLogStream
- logs>DeleteLogGroup
- logs>DeleteLogStream
- logs>DeleteMetricFilter
- logs>DeleteRetentionPolicy
- logs>DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueries
- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetLogEvents
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutRetentionPolicy
- logs:PutSubscriptionFilter
- logs:StartQuery
- logs:StopQuery
- logs:TestMetricFilter
- s3>CreateBucket
- s3>ListBucket
- sns>CreateTopic
- sns:GetTopicAttributes
- sns>ListSubscriptions
- sns>ListTopics
- sns:SetTopicAttributes
- sns:Subscribe
- sns:Unsubscribe
- sqs:GetQueueAttributes
- sqs:GetQueueUrl
- sqs>ListQueues
- sqs:SetQueueAttributes
- swf>CreateAction
- swf:DescribeAction
- swf>ListActionTemplates

- [swf:RegisterAction](#)
- [swf:RegisterDomain](#)
- [swf:UpdateAction](#)

Additionally, to view the service map in ServiceLens, you need [AWSXrayReadOnlyAccess](#)

AWS managed (predefined) policies for CloudWatch

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

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

Note

You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

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

Topics

- [CloudWatchFullAccess \(p. 903\)](#)
- [CloudWatchActionsEC2Access \(p. 904\)](#)
- [CloudWatchReadOnlyAccess \(p. 904\)](#)
- [CloudWatchAutomaticDashboardsAccess \(p. 905\)](#)
- [CloudWatchAgentServerPolicy \(p. 906\)](#)
- [CloudWatchAgentAdminPolicy \(p. 906\)](#)
- [AWS managed \(predefined\) policies for CloudWatch Synthetics \(p. 907\)](#)
- [AWS managed \(predefined\) policies for Amazon CloudWatch RUM \(p. 908\)](#)
- [AWS managed \(predefined\) policies for CloudWatch Evidently \(p. 910\)](#)
- [AWS managed policy for AWS Systems Manager Incident Manager \(p. 912\)](#)

CloudWatchFullAccess

The **CloudWatchFullAccess** policy grants full access to all CloudWatch actions and resources. Its contents are as follows:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "autoscaling:Describe*",  
                "cloudwatch:*",  
                "logs:*",  
                "sns:*",  
                "iam:GetPolicy",  
                "iam:GetPolicyVersion",  
                "iam:GetRole"  
            ]  
        }  
    ]  
}
```

```

        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
        "Condition": {
            "StringLike": {
                "iam:AWSServiceName": "events.amazonaws.com"
            }
        }
    }
]
}

```

CloudWatchActionsEC2Access

The **CloudWatchActionsEC2Access** policy grants read-only access to CloudWatch alarms and metrics in addition to Amazon EC2 metadata. It also grants access to the Stop, Terminate, and Reboot API actions for EC2 instances.

The following is the content of the **CloudWatchActionsEC2Access** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Describe*",
                "ec2:Describe*",
                "ec2:RebootInstances",
                "ec2:StopInstances",
                "ec2:TerminateInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

CloudWatchReadOnlyAccess

The **CloudWatchReadOnlyAccess** policy grants read-only access to CloudWatch.

The following is the content of the **CloudWatchReadOnlyAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Get*",
                "cloudwatch>List*",
                "logs:Get*",
                "logs>List*",
                "logs:StartQuery",
                "logs:StopQuery",

```

```
        "logs:Describe*",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "sns:Get*",
        "sns>List*"
    ],
    "Effect": "Allow",
    "Resource": "*"
}
]
```

CloudWatchAutomaticDashboardsAccess

The **CloudWatch-CrossAccountAccess** managed policy is used by the **CloudWatch-CrossAccountSharingRole** IAM role. This role and policy enable users of cross-account dashboards to view automatic dashboards in each account that is sharing dashboards.

The following is the content of **CloudWatchAutomaticDashboardsAccess**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "cloudfront:GetDistribution",
                "cloudfront>ListDistributions",
                "dynamodb:DescribeTable",
                "dynamodb>ListTables",
                "ec2:DescribeInstances",
                "ec2:DescribeVolumes",
                "ecs:DescribeClusters",
                "ecs:DescribeContainerInstances",
                "ecs>ListClusters",
                "ecs>ListContainerInstances",
                "ecs>ListServices",
                "elasticache:DescribeCacheClusters",
                "elasticbeanstalk:DescribeEnvironments",
                "elasticfilesystem:DescribeFileSystems",
                "elasticloadbalancing:DescribeLoadBalancers",
                "kinesis:DescribeStream",
                "kinesis>ListStreams",
                "lambda:GetFunction",
                "lambda>ListFunctions",
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances",
                "resource-groups>ListGroupResources",
                "resource-groups>ListGroups",
                "route53:GetHealthCheck",
                "route53>ListHealthChecks",
                "s3>ListAllMyBuckets",
                "s3:ListBucket",
                "sns:ListTopics",
                "sns:GetQueueAttributes",
                "sns:GetQueueUrl",
                "sns>ListQueues",
                "synthetics:DescribeCanariesLastRun",
                "tag:GetResources"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {

```

```
        "Action": [
            "apigateway:GET"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:apigateway:*:::/restapis*"
        ]
    }
]
```

CloudWatchAgentServerPolicy

The **CloudWatchAgentServerPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances to allow the CloudWatch agent to read information from the instance and write it to CloudWatch.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ec2:DescribeVolumes",
                "ec2:DescribeTags",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs>CreateLogStream",
                "logs>CreateLogGroup"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": "arn:aws:ssm:parameter/AmazonCloudWatch-*"
        }
    ]
}
```

CloudWatchAgentAdminPolicy

The **CloudWatchAgentAdminPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances. This policy allows the CloudWatch agent to read information from the instance and write it to CloudWatch, and also to write information to Parameter Store.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ec2:DescribeTags",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs>CreateLogStream",
                "logs>CreateLogGroup"
            ],
            "Resource": "*"
        }
    ]
}
```

```

        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm:GetParameter",
            "ssm:PutParameter"
        ],
        "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
    }
]
}

```

Note

You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

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

AWS managed (predefined) policies for CloudWatch Synthetics

The **CloudWatchSyntheticsFullAccess** and **CloudWatchSyntheticsReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch Synthetics. The following additional policies are also relevant:

- **AmazonS3ReadOnlyAccess** and **CloudWatchReadOnlyAccess** – These are necessary to be able to read all Synthetics data in the CloudWatch console.
- **AWSLambdaReadOnlyAccess** – To be able to view the source code used by canaries.
- **CloudWatchSyntheticsFullAccess** enables you to create canaries. Additionally, to create a canary that will have a new IAM role created for it, you also need the following inline policy statement:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}

```

Important

Granting a user the `iam:CreateRole`, `iam:CreatePolicy`, and `iam:AttachRolePolicy` permissions gives that user full administrative access to your AWS account. For example, a user with these permissions can create a policy that has full permissions for all resources, and attach that policy to any role. Be very careful about who you grant these permissions to.

For information about attaching policies and granting permissions to users, see [Changing Permissions for an IAM User](#) and [To embed an inline policy for a user or role](#).

AWS managed (predefined) policies for Amazon CloudWatch RUM

The **AmazonCloudWatchRUMFullAccess** and **AmazonCloudWatchRUMReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch RUM.

AmazonCloudWatchRUMFullAccess

The following are the contents of the **AmazonCloudWatchRUMFullAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rum:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam>CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/rum.amazonaws.com/
AWSServiceRoleForRealUserMonitoring"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/RUM-Monitor*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "cognito-identity.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch>ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "arn:aws:cloudwatch:*:alarm:*
        },
        {
            "Effect": "Allow",
            "
```

```

    "Action": [
        "cognito-identity>CreateIdentityPool",
        "cognito-identity>ListIdentityPools",
        "cognito-identity>DescribeIdentityPool",
        "cognito-identity>GetIdentityPoolRoles",
        "cognito-identity>SetIdentityPoolRoles"
    ],
    "Resource": "arn:aws:cognito-identity:*::identitypool/*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogGroup",
        "logs>DeleteLogGroup",
        "logs>PutRetentionPolicy",
        "logs>CreateLogStream"
    ],
    "Resource": "arn:aws:logs::*:log-group:*RUMService*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogDelivery",
        "logs>GetLogDelivery",
        "logs>UpdateLogDelivery",
        "logs>DeleteLogDelivery",
        "logs>ListLogDeliveries",
        "logs>DescribeResourcePolicies"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs>DescribeLogGroups"
    ],
    "Resource": "arn:aws:logs::*:log-group::log-stream:/*"
},
{
    "Effect": "Allow",
    "Action": [
        "synthetics>describeCanaries",
        "synthetics>describeCanariesLastRun"
    ],
    "Resource": "arn:aws:synthetics::*:canary:/*"
}
]
}

```

[AmazonCloudWatchRUMReadOnlyAccess](#)

The following are the contents of the **AmazonCloudWatchRUMReadOnlyAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rum:Get*",
                "rum>List*"
            ],
            "Resource": "*"
        }
    ]
}
```

```
    ]  
}
```

AmazonCloudWatchRUMServiceRolePolicy

You can't attach **AmazonCloudWatchRUMServiceRolePolicy** to your IAM entities. This policy is attached to a service-linked role that allows CloudWatch RUM to publish monitoring data to other relevant AWS services. For more information about this service linked role, see [Using service-linked roles for CloudWatch RUM \(p. 926\)](#).

AWS managed (predefined) policies for CloudWatch Evidently

The **CloudWatchEvidentlyFullAccess** and **CloudWatchEvidentlyReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch Evidently.

CloudWatchEvidentlyFullAccess

The following are the contents of the **CloudWatchEvidentlyFullAccess** policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "evidently:*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListRoles"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetRole"  
            ],  
            "Resource": [  
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetBucketLocation",  
                "s3>ListAllMyBuckets"  
            ],  
            "Resource": "arn:aws:s3:::/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:GetMetricData",  
                "cloudwatch:GetMetricStatistics",  
                "cloudwatchDescribeAlarmHistory",  
                "cloudwatchDescribeAlarmsForMetric",  
                "cloudwatchListTagsForResource"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```

},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:DescribeAlarms",
        "cloudwatch:TagResource",
        "cloudwatch:UnTagResource"
    ],
    "Resource": [
        "arn:aws:cloudwatch:*::alarm:/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudtrail:LookupEvents"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricAlarm"
    ],
    "Resource": [
        "arn:aws:cloudwatch::*:alarm:Evidently-Alarm-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns>ListTopics"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns>CreateTopic",
        "sns:Subscribe",
        "sns>ListSubscriptionsByTopic"
    ],
    "Resource": [
        "arn:*:sns:*:Evidently-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

[CloudWatchEvidentlyReadOnlyAccess](#)

The following are the contents of the **CloudWatchEvidentlyReadOnlyAccess** policy.

```
{
}
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:GetExperiment",
                "evidently:GetFeature",
                "evidently:GetLaunch",
                "evidently:GetProject",
                "evidently>ListExperiments",
                "evidently>ListFeatures",
                "evidently>ListLaunches",
                "evidently>ListProjects"
            ],
            "Resource": "*"
        }
    ]
}

```

AWS managed policy for AWS Systems Manager Incident Manager

The **AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy** policy is attached to a service-linked role that allows CloudWatch to start incidents in AWS Systems Manager Incident Manager on your behalf. For more information, see [Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions \(p. 923\)](#).

The policy has the following permission:

- ssm-incidents:StartIncident

CloudWatchSyntheticsFullAccess

The following is the content of the **CloudWatchSyntheticsFullAccess** policy.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::cw-syn-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam>ListRoles",
                "s3>ListAllMyBuckets",
                "xray:GetTraceSummaries",
                "xray:BatchGetTraces",
                "apigateway:GET"
            ]
        }
    ]
}

```

```

        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation"
        ],
        "Resource": "arn:aws:s3:::arn:aws:s3:::cw-syn-*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3>ListBucket"
        ],
        "Resource": "arn:aws:s3:::arn:aws:s3:::aws-synthetics-library-*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": [
            "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
        ],
        "Condition": {
            "StringEquals": {
                "iam:PassedToService": [
                    "lambda.amazonaws.com",
                    "synthetics.amazonaws.com"
                ]
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole"
        ],
        "Resource": [
            "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:GetMetricStatistics"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "cloudwatch:PutMetricAlarm",
            "cloudwatch>DeleteAlarms"
        ],
        "Resource": [

```

```
        "arn:aws:cloudwatch:*::*:alarm:Synthetics-*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:DescribeAlarms"
    ],
    "Resource": [
        "arn:aws:cloudwatch:*::*:alarm:/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "lambda>CreateFunction",
        "lambda>AddPermission",
        "lambda>PublishVersion",
        "lambda>UpdateFunctionCode",
        "lambda>UpdateFunctionConfiguration",
        "lambda>GetFunctionConfiguration"
    ],
    "Resource": [
        "arn:aws:lambda:*::*:function:cwsyn-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "lambda>GetLayerVersion",
        "lambda>PublishLayerVersion"
    ],
    "Resource": [
        "arn:aws:lambda:*::*:layer:cwsyn-*",
        "arn:aws:lambda:*::*:layer:Synthetics:/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2>DescribeVpcs",
        "ec2>DescribeSubnets",
        "ec2>DescribeSecurityGroups"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns>ListTopics"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns>CreateTopic",
        "sns>Subscribe",
        "sns>ListSubscriptionsByTopic"
    ],
    "Resource": [
        "arn:*:sns:*::*:Synthetics-*"
    ]
}
```

```

        ],
    },
    {
        "Effect": "Allow",
        "Action": [
            "kms>ListAliases"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kms>DescribeKey"
        ],
        "Resource": "arn:aws:kms:*::key/*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kms>Decrypt"
        ],
        "Resource": "arn:aws:kms:*::key/*",
        "Condition": {
            "StringLike": {
                "kms>ViaService": [
                    "s3.*.amazonaws.com"
                ]
            }
        }
    }
]
}

```

[CloudWatchSyntheticsReadOnlyAccess](#)

The following is the content of the **CloudWatchSyntheticsReadOnlyAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics>Describe*",
                "synthetics>Get*",
                "synthetics>List*"
            ],
            "Resource": "*"
        }
    ]
}
```

[Customer managed policy examples](#)

In this section, you can find example user policies that grant permissions for various CloudWatch actions. These policies work when you are using the CloudWatch API, AWS SDKs, or the AWS CLI.

Examples

- [Example 1: Allow user full access to CloudWatch \(p. 916\)](#)
- [Example 2: Allow read-only access to CloudWatch \(p. 916\)](#)
- [Example 3: Stop or terminate an Amazon EC2 instance \(p. 916\)](#)

Example 1: Allow user full access to CloudWatch

To grant a user full access to CloudWatch, you can use grant them the **CloudWatchFullAccess** managed policy instead of creating a customer-managed policy. The contents of the **CloudWatchFullAccess** are listed in [CloudWatchFullAccess \(p. 903\)](#).

Example 2: Allow read-only access to CloudWatch

The following policy allows a user read-only access to CloudWatch and view Amazon EC2 Auto Scaling actions, CloudWatch metrics, CloudWatch Logs data, and alarm-related Amazon SNS data.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "autoscaling:Describe*",  
                "cloudwatch:Describe*",  
                "cloudwatch:Get*",  
                "cloudwatch>List*",  
                "logs:Get*",  
                "logs:Describe*",  
                "sns:Get*",  
                "sns>List*"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

Example 3: Stop or terminate an Amazon EC2 instance

The following policy allows an CloudWatch alarm action to stop or terminate an EC2 instance. In the sample below, the GetMetricData, ListMetrics, and DescribeAlarms actions are optional. It is recommended that you include these actions to ensure that you have correctly stopped or terminated the instance.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "cloudwatch:PutMetricAlarm",  
                "cloudwatch:DescribeAlarms",  
                "cloudwatch:ListMetrics",  
                "cloudwatch:GetMetricData"  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Effect": "Allow"  
        },  
        {  
            "Action": [  
                "ec2:DescribeInstances",  
                "ec2:StopInstances",  
                "ec2:TerminateInstances"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

```

        ],
        "Effect": "Allow"
    }
}
]
```

CloudWatch updates to AWS managed policies

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

Change	Description	Date
AmazonCloudWatchRUMFullAccess (p. 776) – New policy	<p>CloudWatch added a new policy to enable full management of CloudWatch RUM.</p> <p>CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see Use CloudWatch RUM (p. 776).</p>	November 29, 2021
AmazonCloudWatchRUMReadOnly (p. 776) – New policy	<p>CloudWatch added a new policy to enable read-only access to CloudWatch RUM.</p> <p>CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see Use CloudWatch RUM (p. 776).</p>	November 29, 2021
CloudWatchEvidentlyFullAccess (p. 808) – New policy	<p>CloudWatch added a new policy to enable full management of CloudWatch Evidently.</p> <p>CloudWatch Evidently allows you to perform A/B experiments of your web applications, and to roll them out gradually. For more information, see Perform launches and A/B experiments with CloudWatch Evidently (p. 808).</p>	November 29, 2021
CloudWatchEvidentlyReadOnlyAccess (p. 808) – New policy	<p>CloudWatch added a new policy to enable read-only access to CloudWatch Evidently.</p> <p>CloudWatch Evidently allows you to perform A/B experiments of your web applications, and to roll them out gradually. For more information, see Perform launches and A/B experiments with CloudWatch Evidently (p. 808).</p>	November 29, 2021

Change	Description	Date
	Perform launches and A/B experiments with CloudWatch Evidently (p. 808).	
AWSServiceRoleForCloudWatchRUM (p. 808) – New managed policy	CloudWatch added a policy for a new service-linked role to allow CloudWatch RUM to publish monitoring data to other relevant AWS services.	November 29, 2021
CloudWatchSyntheticsFullAccess (p. 808) – Update to an existing policy	<p>CloudWatch Synthetics added permissions to CloudWatchSyntheticsFullAccess, and also changed the scope of one permission.</p> <p>The <code>kms>ListAliases</code> permission was added so that users can list available AWS KMS keys that can be used to encrypt canary artifacts. The <code>kmsDescribeKey</code> permission was added so that users can see the details of keys that will be used to encrypt for canary artifacts. And the <code>kmsDecrypt</code> permission was added to enable users to decrypt canary artifacts. This decryption ability is limited to use on resources within Amazon S3 buckets.</p> <p>The Resource scope of the <code>s3GetBucketLocation</code> permission was changed from * to <code>arn:aws:s3:::*</code>.</p>	September 29, 2021
CloudWatchSyntheticsFullAccess (p. 808) – Update to an existing policy	<p>CloudWatch Synthetics added a permission to CloudWatchSyntheticsFullAccess.</p> <p>The <code>lambdaUpdateFunctionCode</code> permission was added so that users with this policy can change the runtime version of canaries.</p>	July 20, 2021
AWSCloudWatchAlarms_ActionSSM (p. 912) – New managed policy	<p>CloudWatch added a new managed IAM policy to allow CloudWatch to create incidents in AWS Systems Manager Incident Manager.</p>	May 10, 2021

Change	Description	Date
CloudWatchAutomaticDashboards – Update to an existing policy	CloudWatch added a permission to the CloudWatchAutomaticDashboardsAccess managed policy. The <code>synthetics:DescribeCanariesLastRun</code> permission was added to this policy to enable cross-account dashboard users to see details about CloudWatch Synthetics canary runs.	April 20, 2021
CloudWatch started tracking changes	CloudWatch started tracking changes for its AWS managed policies.	April 14, 2021

Using condition keys to limit access to CloudWatch namespaces

Use IAM condition keys to limit users to publishing metrics only in the CloudWatch namespaces that you specify.

Allowing publishing in one namespace only

The following policy limits the user to publishing metrics only in the namespace named `MyCustomNamespace`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "MyCustomNamespace"
                }
            }
        }
    ]
}
```

Excluding publishing from a namespace

The following policy allows the user to publish metrics in any namespace except for `CustomNamespace2`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData"
        },
        {
            "Effect": "Deny",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringNotEquals": {
                    "cloudwatch:namespace": "CustomNamespace2"
                }
            }
        }
    ]
}
```

```
        "Effect": "Deny",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "CustomNamespace2"
            }
        }
    ]
}
```

Using condition keys to limit Contributor Insights users' access to log groups

To create a rule in Contributor Insights and see its results, a user must have the `cloudwatch:PutInsightRule` permission. By default, a user with this permission can create a Contributor Insights rule that evaluates any log group in CloudWatch Logs and then see the results. The results can contain contributor data for those log groups.

You can create IAM policies with condition keys to grant users the permission to write Contributor Insights rules for some log groups while preventing them from writing rules for and seeing this data from other log groups.

For more information about the `Condition` element in IAM policies, see [IAM JSON policy elements: Condition](#).

Allow access to write rules and view results for only certain log groups

The following policy allows the user access to write rules and view results for the log group named `AllowedLogGroup` and all log groups that have names that start with `AllowedWildcard`. It does not grant access to write rules or view rule results for any other log groups.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCertainLogGroups",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "AllowedLogGroup",
                        "AllowedWildcard*"
                    ]
                }
            }
        ]
    ]
}
```

Deny writing rules for specific log groups but allow writing rules for all other log groups

The following policy explicitly denies the user access to write rules and view rule results for the log group named `ExplicitlyDeniedLogGroup`, but allows writing rules and viewing rule results for all other log groups.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInsightRulesOnLogGroupsByDefault",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:***:insight-rule/*"
        },
        {
            "Sid": "ExplicitDenySomeLogGroups",
            "Effect": "Deny",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:***:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "/test/alpine/ExplicitlyDeniedLogGroup"
                    ]
                }
            }
        }
    ]
}
```

Using condition keys to limit alarm actions

When CloudWatch alarms change state, they can perform different actions such as stopping and terminating EC2 instances and performing Systems Manager actions. These actions can be initiated when the alarm changes to any state, including ALARM, OK, or INSUFFICIENT_DATA.

Use the `cloudwatch:AlarmActions` condition key to allow a user to create alarms that can only perform the actions you specify when the alarm state changes. For example, you can allow a user to create alarms that can only perform actions which are not EC2 actions.

Allow a user to create alarms that can only send Amazon SNS notifications or perform Systems Manager actions

The following policy limits the user to creating alarms that can only send Amazon SNS notifications and perform Systems Manager actions. The user can't create alarms that perform EC2 actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAlarmsThatCanPerformOnlySNSandSSMActions",
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricAlarm",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "cloudwatch:AlarmActions": [
                        "arn:aws:sns:*",
                        "arn:aws:ssm:)"
                    ]
                }
            }
        }
    ]
}
```

Using service-linked roles for CloudWatch

Amazon CloudWatch 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 CloudWatch. Service-linked roles are predefined by CloudWatch and include all the permissions that the service requires to call other AWS services on your behalf.

One service-linked role in CloudWatch makes setting up CloudWatch alarms that can terminate, stop, or reboot an Amazon EC2 instance without requiring you to manually add the necessary permissions. Another service-linked role enables a monitoring account to access CloudWatch data from other accounts that you specify, to build cross-account cross-Region dashboards.

CloudWatch defines the permissions of these service-linked roles, and unless defined otherwise, only CloudWatch can assume the role. 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 the roles only after first deleting their related resources. This restriction protects your CloudWatch resources because you can't inadvertently remove permissions 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 CloudWatch alarms EC2 actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchEvents** – CloudWatch uses this service-linked role to perform Amazon EC2 alarm actions.

The **AWSServiceRoleForCloudWatchEvents** service-linked role trusts the CloudWatch Events service to assume the role. CloudWatch Events invokes the terminate, stop, or reboot instance actions when called upon by the alarm.

The **AWSServiceRoleForCloudWatchEvents** service-linked role permissions policy allows CloudWatch Events to complete the following actions on Amazon EC2 instances:

- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RecoverInstances`
- `ec2:DescribeInstanceRecoveryAttribute`
- `ec2:DescribeInstances`
- `ec2:DescribeInstanceStatus`

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

Service-linked role permissions for CloudWatch alarms Systems Manager OpsCenter actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchAlarms_ActionSSM** – CloudWatch uses this service-linked role to perform Systems Manager OpsCenter actions when a CloudWatch alarm goes into ALARM state.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSM** service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager OpsCenter actions when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSM** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm:CreateOpsItem`

Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions

CloudWatch uses the service-linked role named

AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents – CloudWatch uses this service-linked role to start Incident Manager incidents when a CloudWatch alarm goes into ALARM state.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager Incident Manager action when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm-incidents:StartIncident`

Service-linked role permissions for CloudWatch cross-account cross-Region

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchCrossAccount** – CloudWatch uses this role to access CloudWatch data in other AWS accounts that you specify. The SLR only provides the assume role permission to allow the CloudWatch service to assume the role in the sharing account. It is the sharing role that provides access to data.

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role trusts the CloudWatch service to assume the role.

Creating a service-linked role for CloudWatch

You do not need to manually create any of these service-linked roles. The first time you create an alarm in the AWS Management Console, the IAM CLI, or the IAM API, CloudWatch creates **AWSServiceRoleForCloudWatchEvents** and **AWSServiceRoleForCloudWatchAlarms_ActionSSM** for you. The first time When you first enable an account to be a monitoring account for cross-account cross-Region functionality, CloudWatch creates **AWSServiceRoleForCloudWatchCrossAccount** for you.

For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*.

Editing a service-linked role for CloudWatch

CloudWatch does not allow you to edit the **AWSServiceRoleForCloudWatchEvents**, **AWSServiceRoleForCloudWatchAlarms_ActionSSM** or **AWSServiceRoleForCloudWatchCrossAccount**

roles. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, use the following command:

```
$ aws iam get-role --role-name role-name
```

Use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **myrole**.

2. To update a service-linked role's description, use the following command:

```
$ aws iam update-role-description --role-name role-name --description description
```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

[GetRole](#)

2. To update a role's description, use the following command:

[UpdateRoleDescription](#)

Deleting a service-linked role for CloudWatch

If you no longer have alarms that automatically stop, terminate, or reboot EC2 instances, we recommend that you delete the `AWSServiceRoleForCloudWatchEvents` role.

If you no longer have alarms that perform Systems Manager OpsCenter actions, we recommend that you delete the `AWSServiceRoleForCloudWatchAlarms_ActionSSM` role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the AWSServiceRoleForCloudWatchEvents role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

Note

If you are unsure whether CloudWatch is using the AWSServiceRoleForCloudWatchEvents role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call [DeleteServiceLinkedRole](#). In the request, specify the role name that you want to delete.

Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call [GetServiceLinkedRoleDeletionStatus](#). In the request, specify the `DeletionTaskId`.

The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

CloudWatch updates to AWS service-linked roles

View details about updates to AWS managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [CloudWatch Document history page](#).

Change	Description	Date
AWSServiceRoleForCloudWatchAlarms – New service-linked role	CloudWatch added a new (p. 923) service-linked role to allow CloudWatch to create incidents in AWS Systems Manager Incident Manager.	April 26, 2021
CloudWatch started tracking changes	CloudWatch started tracking changes for its service linked roles.	April 26, 2021

Using service-linked roles for CloudWatch RUM

CloudWatch RUM 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 RUM. Service-linked roles are predefined by RUM and include all the permissions that the service requires to call other AWS services on your behalf.

RUM defines the permissions of these service-linked roles, and unless defined otherwise, only RUM can assume the role. 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 the roles only after first deleting their related resources. This restriction protects your RUM resources because you can't inadvertently remove permissions 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 RUM

RUM uses the service-linked role named **AWSServiceRoleForCloudWatchRUM** – this role allows RUM to send AWS X-Ray trace data into your account, for app monitors that you enable X-Ray tracing for.

The **AWSServiceRoleForCloudWatchRUM** service-linked role trusts the X-Ray service to assume the role. X-Ray sends the trace data to your account.

The **AWSServiceRoleForCloudWatchRUM** service-linked role has an IAM policy attached named **AmazonCloudWatchRUMServiceRolePolicy**. This policy grants permission to CloudWatch RUM to publish monitoring data to other relevant AWS service. It includes permissions that allow RUM to complete the following actions:

- `xray:PutTraceSegments`

Creating a service-linked role for RUM

You do not need to manually create the service-linked role for CloudWatch RUM. The first time that you create an app monitor with X-Ray tracing enabled, or update an app monitor to use X-Ray tracing, RUM creates **AWSServiceRoleForCloudWatchRUM** for you.

For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*.

Editing a service-linked role for RUM

CloudWatch RUM does not allow you to edit the **AWSServiceRoleForCloudWatchRUM** role. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, use the following commands:

```
$ aws iam get-role --role-name role-name
```

Use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: arn:aws:iam::123456789012:role/myrole, you refer to the role as **myrole**.

2. To update a service-linked role's description, use the following command:

```
$ aws iam update-role-description --role-name role-name --description description
```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

[GetRole](#)

2. To update a role's description, use the following command:

[UpdateRoleDescription](#)

Deleting a service-linked role for RUM

If you no longer have app monitors with X-Ray enabled, we recommend that you delete the **AWSServiceRoleForCloudWatchRUM** role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the **AWSServiceRoleForCloudWatchRUM** role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

Note

If you are unsure whether RUM is using the **AWSServiceRoleForCloudWatchRUM** role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call `DeleteServiceLinkedRole`. In the request, specify the role name that you want to delete.

Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call `GetServiceLinkedRoleDeletionStatus`. In the request, specify the `DeletionTaskId`.

The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Using service-linked roles for CloudWatch Application Insights

CloudWatch Application Insights 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 CloudWatch Application Insights. Service-linked roles are predefined by CloudWatch Application Insights and include all of the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CloudWatch Application Insights easier because you don't have to manually add the necessary permissions. CloudWatch Application Insights defines the permissions of its service-linked roles, and unless defined otherwise, only CloudWatch Application Insights 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.

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** link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch Application Insights

CloudWatch Application Insights uses the service-linked role named **AWSServiceRoleForApplicationInsights**. Application Insights uses this role to perform operations such as analyzing the resource groups of the customer, creating CloudFormation stacks to create alarms on metrics, and configuring the CloudWatch Agent on EC2 instances. This service-linked role is attached to the following managed policy: `CloudwatchApplicationInsightsServiceLinkedRolePolicy`. For updates to this policy, see [Application Insights updates to AWS managed policies \(p. 940\)](#).

The role permissions policy allows CloudWatch Application Insights to complete the following actions on resources.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:DescribeAlarmHistory",  
                "cloudwatch:DescribeAlarms",  
                "cloudwatch:GetMetricData",  
                "cloudwatch>ListMetrics",  
                "cloudwatch:PutMetricAlarm",  
                "cloudwatch>DeleteAlarms",  
                "cloudwatch:PutAnomalyDetector",  
                "cloudwatch>DeleteAnomalyDetector",  
                "cloudwatch:DescribeAnomalyDetectors"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:FilterLogEvents",  
                "logs:GetLogEvents",  
                "logs:DescribeLogStreams",  
                "logs:DescribeLogGroups"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

```
    "Resource": [
        "*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "events:DescribeRule"
    ],
    "Resource": [
        "*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "cloudFormation>CreateStack",
        "cloudFormation>UpdateStack",
        "cloudFormation>DeleteStack",
        "cloudFormation>DescribeStackResources"
    ],
    "Resource": [
        "arn:aws:cloudformation:*.*:stack/ApplicationInsights-*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "cloudFormation>DescribeStacks",
        "cloudFormation>ListStackResources"
    ],
    "Resource": [
        "*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "tag:GetResources"
    ],
    "Resource": [
        "*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "resource-groups>ListGroupResources",
        "resource-groups>GetGroupQuery",
        "resource-groups>GetGroup"
    ],
    "Resource": [
        "*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "resource-groups>CreateGroup",
        "resource-groups>DeleteGroup"
    ],
    "Resource": [
        "arn:aws:resource-groups:*.*:group/ApplicationInsights-*"
    ],
},
```

```

        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:DescribeLoadBalancers",
            "elasticloadbalancing:DescribeTargetGroups",
            "elasticloadbalancing:DescribeTargetHealth"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "autoscaling:DescribeAutoScalingGroups"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm:PutParameter",
            "ssm:DeleteParameter",
            "ssm:AddTagsToResource",
            "ssm:RemoveTagsFromResource",
            "ssm:GetParameters"
        ],
        "Resource": "arn:aws:ssm:*::parameter/AmazonCloudWatch-ApplicationInsights-*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm>CreateAssociation",
            "ssm:UpdateAssociation",
            "ssm:DeleteAssociation",
            "ssm:DescribeAssociation"
        ],
        "Resource": [
            "arn:aws:ec2::*:instance/*",
            "arn:aws:ssm::*:association/*",
            "arn:aws:ssm::*:managed-instance/*",
            "arn:aws:ssm::*:document/AWSEC2-
ApplicationInsightsCloudwatchAgentInstallAndConfigure",
            "arn:aws:ssm::*:document/AWS-ConfigureAWSPackage",
            "arn:aws:ssm::*:document/AmazonCloudWatch-ManageAgent"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm:GetOpsItem",
            "ssm:CreateOpsItem",
            "ssm:DescribeOpsItems",
            "ssm:UpdateOpsItem",
            "ssm:DescribeInstanceInformation"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm:AddTagsToResource"
        ],

```

```
        "Resource": "arn:aws:ssm:*.*:opsitem/*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm>ListCommandInvocations"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": "ssm:SendCommand",
        "Resource": [
            "arn:aws:ec2:*.*:instance/*",
            "arn:aws:ssm:*.*:document/AWSEC2-CheckPerformanceCounterSets",
            "arn:aws:ssm:*.*:document/AWS-ConfigureAWSPackage",
            "arn:aws:ssm:*.*:document/AWSEC2-DetectWorkload",
            "arn:aws:ssm:*.*:document/AmazonCloudWatch-ManageAgent"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeInstances",
            "ec2:DescribeVolumes",
            "ec2:DescribeVolumeStatus"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DescribeDBInstances",
            "rds:DescribeDBClusters"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lambda>ListFunctions",
            "lambda:GetFunctionConfiguration",
            "lambda>ListEventSourceMappings"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "events:PutRule",
            "events:PutTargets",
            "events:RemoveTargets",
            "events:DeleteRule"
        ],
        "Resource": [
            "arn:aws:events:*.*:rule/AmazonCloudWatch-ApplicationInsights-*"
        ]
    },
}
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "xray:GetServiceGraph",  
        "xray:GetTraceSummaries",  
        "xray:GetTimeSeriesServiceStatistics",  
        "xray:GetTraceGraph"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "dynamodb>ListTables",  
        "dynamodb>DescribeTable",  
        "dynamodb>DescribeContributorInsights",  
        "dynamodb>DescribeTimeToLive"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "application-autoscaling>DescribeScalableTargets"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "s3>ListAllMyBuckets",  
        "s3>GetMetricsConfiguration",  
        "s3>GetReplicationConfiguration"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "states>ListStateMachine",  
        "states>DescribeExecution",  
        "states>DescribeStateMachine",  
        "states>GetExecutionHistory"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "apigateway:GET"  
    ],  
    "Resource": [  
        "*"  
    ]  
},  
{
```

```
"Effect": "Allow",
"Action": [
    "ecs:DescribeClusters",
    "ecs:DescribeContainerInstances",
    "ecs:DescribeServices",
    "ecs:DescribeTaskDefinition",
    "ecs:DescribeTasks",
    "ecs:DescribeTaskSets",
    "ecs>ListClusters",
    "ecs>ListContainerInstances",
    "ecs>ListServices",
    "ecs>ListTasks"
],
"Resource": [
    "*"
],
},
{
    "Effect": "Allow",
    "Action": [
        "ecs:UpdateClusterSettings"
    ],
    "Resource": [
        "arn:aws:ecs:*::cluster/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "eks:DescribeCluster",
        "eks:DescribeFargateProfile",
        "eks:DescribeNodegroup",
        "eks>ListClusters",
        "eks>ListFargateProfiles",
        "eks>ListNodegroups",
        "fsx:DescribeFileSystems"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns:GetSubscriptionAttributes",
        "sns:GetTopicAttributes",
        "sns:GetSMSAttributes",
        "sns>ListSubscriptionsByTopic",
        "sns>ListTopics"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns:ListQueues"
    ],
    "Resource": "*"
}
{
    "Effect": "Allow",
    "Action": [
        "logs>DeleteSubscriptionFilter"
    ],

```

```
    "Resource": [
        "arn:aws:logs:*:log-group:*
```

```
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "logs:PutSubscriptionFilter"
    ],
    "Resource": [
        "arn:aws:logs:*:log-group:*,"
        "arn:aws:logs:*:destination:AmazonCloudWatch-ApplicationInsights-
LogIngestionDestination*"
    ]
}
]
```

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 CloudWatch Application Insights

You don't need to manually create a service-linked role. When you create a new Application Insights application in the AWS Management Console, CloudWatch Application Insights creates the service-linked role for you.

If you delete this service-linked role, and then want to create it again, you can use the same process to recreate the role in your account. When you create a new Application Insights application, CloudWatch Application Insights creates the service-linked role for you again.

Editing a service-linked role for CloudWatch Application Insights

CloudWatch Application Insights does not allow you to edit the AWSServiceRoleForApplicationInsights 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 CloudWatch Application Insights

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 avoid having an unused entity that is not actively monitored or maintained. However, you must delete all applications in Application Insights before you can manually delete the role.

Note

If the CloudWatch Application Insights 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 CloudWatch Application Insights resources used by the AWSServiceRoleForApplicationInsights

- Delete all of your CloudWatch Application Insights applications. For more information, see "Deleting Your Application(s)" in the CloudWatch Application Insights User Guide.

To manually delete the service-linked role using IAM

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

Supported Regions for CloudWatch Application Insights service-linked roles

CloudWatch Application Insights supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see [CloudWatch Application Insights Regions and Endpoints](#).

AWS managed policies for Amazon CloudWatch Application Insights

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 `ViewOnlyAccess` AWS managed policy provides read-only access to many 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*.

Managed policies

- [AWS managed policy: CloudWatchApplicationInsightsFullAccess \(p. 937\)](#)
- [AWS managed policy: CloudWatchApplicationInsightsReadOnlyAccess \(p. 939\)](#)
- [AWS managed policy: CloudwatchApplicationInsightsServiceLinkedRolePolicy \(p. 939\)](#)
- [Application Insights updates to AWS managed policies \(p. 940\)](#)

Managed policies

- [AWS managed policy: CloudWatchApplicationInsightsFullAccess \(p. 937\)](#)
- [AWS managed policy: CloudWatchApplicationInsightsReadOnlyAccess \(p. 939\)](#)
- [AWS managed policy: CloudwatchApplicationInsightsServiceLinkedRolePolicy \(p. 939\)](#)
- [Application Insights updates to AWS managed policies \(p. 940\)](#)

AWS managed policy: CloudWatchApplicationInsightsFullAccess

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

This policy grants administrative permissions that allow full access to Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows full access to Application Insights functionality.
- `iam` – Allows Application Insights to create the service-linked role, `AWSServiceRoleForApplicationInsights`. This is required so that Application Insights can perform operations such as analyze the resource groups of a customer, create CloudFormation stacks to create alarms on metrics, and configure the CloudWatch Agent on EC2 instances. For more information, see [Using service-linked roles for CloudWatch Application Insights \(p. 930\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "applicationinsights:*",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeInstances",  
                "ec2:DescribeVolumes",  
                "rds:DescribeDBInstances",  
                "rds:DescribeDBClusters",  
                "sns>ListQueues",  
                "elasticloadbalancing:DescribeLoadBalancers",  
                "elasticloadbalancing:DescribeTargetGroups",  
                "elasticloadbalancing:DescribeTargetHealth",  
                "autoscaling:DescribeAutoScalingGroups",  
                "lambda>ListFunctions",  
                "dynamodb>ListTables",  
                "s3>ListAllMyBuckets",  
                "sns>ListTopics",  
                "states>ListStateMachines",  
                "apigateway:GET",  
                "ecs>ListClusters",  
                "ecs:DescribeTaskDefinition",  
                "ecs>ListServices",  
                "ecs>ListTasks",  
                "eks>ListClusters",  
                "eks>ListNodegroups",  
                "fsx:DescribeFileSystems",  
                "logs:DescribeLogGroups"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam>CreateServiceLinkedRole"  
            ]  
        }  
    ]  
}
```

```
        ],
        "Resource": [
            "arn:aws:iam::*:role/aws-service-role/application-insights.amazonaws.com/
AWSServiceRoleForApplicationInsights"
        ],
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "application-insights.amazonaws.com"
            }
        }
    ]
}
```

AWS managed policy: CloudWatchApplicationInsightsReadOnlyAccess

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

This policy grants administrative permissions that allow read-only access to all Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows read-only access to Application Insights functionality.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "applicationinsights:Describe*",
                "applicationinsights>List*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: CloudwatchApplicationInsightsServiceLinkedRolePolicy

You can't attach `CloudwatchApplicationInsightsServiceLinkedRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Application Insights to monitor customer resources. For more information, see [Using service-linked roles for CloudWatch Application Insights \(p. 930\)](#).

Application Insights updates to AWS managed policies

View details about updates to AWS managed policies for Application Insights since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Application Insights [Document history \(p. 969\)](#) page.

Change	Description	Date
CloudWatchApplicationInsightsFull – Update to an existing policy	<p><small>Application Insights added a new permission to describe log groups.</small></p> <p>This permissions is required for Amazon CloudWatch Application Insights to ensure that the correct permissions for monitoring log groups are in an account when creating a new application.</p>	January 24, 2022
CloudwatchApplicationInsightsServerless – Update to an existing policy	<p><small>Application Insights added new permissions to create and delete CloudWatch Log Subscription Filters.</small></p> <p>These permissions are required for Amazon CloudWatch Application Insights to create Subscription Filters to facilitate log monitoring of resources within configured applications.</p>	January 24, 2022
CloudWatchApplicationInsightsFull – Update to an existing policy	<p><small>Application Insights added new permissions to describe target groups and target health for Elastic Load Balancers.</small></p> <p>These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.</p>	November 4, 2021
CloudwatchApplicationInsightsServerless – Update to an existing policy	<p><small>Application Insights added new permissions to run the AmazonCloudWatch-ManageAgent SSM document on Amazon EC2 instances.</small></p> <p>This permissions is required for Amazon CloudWatch Application Insights to clean up CloudWatch agent configuration files created by Application Insights.</p>	September 30, 2021

Change	Description	Date
CloudwatchApplicationInsightsServerPolicy – Update to an existing policy	<p>Application Insights added new permissions to support account-based application monitoring to onboard and monitor all supported resources in your account.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to query, tag resources, and create groups for these resources.</p> <p>Application Insights added new permissions to support monitoring of SNS topics.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to gather metadata from SNS resources to configure monitoring for SNS topics.</p>	September 15, 2021
CloudWatchApplicationInsightsFullAccessPolicy – Update to an existing policy	<p>Application Insights added new permissions to describe and list supported resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.</p>	September 15, 2021
CloudwatchApplicationInsightsServerPolicy – Update to an existing policy	<p>Application Insights added new permissions to describe FSx resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to read customer FSx resource configurations, and to help customers automatically set up best practice FSx monitoring with CloudWatch.</p>	August 31, 2021

Change	Description	Date
CloudwatchApplicationInsightsService – Update to an existing policy	<p>Application Insights added new permissions to describe and list ECS and EKS service resources.</p> <p>This permission is required for Amazon CloudWatch Application Insights to read customer container resources configuration, and to help customers automatically set up best practice container monitoring with CloudWatch.</p>	May 18, 2021
CloudwatchApplicationInsightsService – Update to an existing policy	<p>Application Insights added new permissions to allow OpsCenter to tag OpsItems using the <code>ssm:AddTagsToResource</code> action on resources with the <code>opsitem</code> resource type.</p> <p>This permission is required by OpsCenter. Amazon CloudWatch Application Insights creates OpsItems so that the customer can resolve problems using AWS SSM OpsCenter.</p>	April 13, 2021
Application Insights started tracking changes	Application Insights started tracking changes for its AWS managed policies.	April 13, 2021

Amazon CloudWatch permissions reference

When you are setting up [Access control \(p. 895\)](#) and writing permissions policies that you can attach to an IAM identity (identity-based policies), you can use the following table as a reference. The table lists each CloudWatch API operation and the corresponding actions for which you can grant permissions to perform the action. You specify the actions in the policy's `Action` field, and you specify a wildcard character (*) as the resource value in the policy's `Resource` field.

You can use AWS-wide condition keys in your CloudWatch policies to express conditions. For a complete list of AWS-wide keys, see [AWS Global and IAM Condition Context Keys](#) in the *IAM User Guide*.

Note

To specify an action, use the `cloudwatch:` prefix followed by the API operation name. For example: `cloudwatch:GetMetricData`, `cloudwatch>ListMetrics`, or `cloudwatch:*` (for all CloudWatch actions).

Topics

- [CloudWatch API operations and required permissions for actions \(p. 943\)](#)
- [CloudWatch Contributor Insights API operations and required permissions for actions \(p. 945\)](#)
- [CloudWatch Events API operations and required permissions for actions \(p. 946\)](#)
- [CloudWatch Logs API operations and required permissions for actions \(p. 947\)](#)
- [Amazon EC2 API operations and required permissions for actions \(p. 950\)](#)
- [Amazon EC2 Auto Scaling API operations and required permissions for actions \(p. 950\)](#)

CloudWatch API operations and required permissions for actions

CloudWatch API operations	Required permissions (API actions)
DeleteAlarms	<code>cloudwatch:DeleteAlarms</code> Required to delete an alarm.
DeleteDashboards	<code>cloudwatch:DeleteDashboards</code> Required to delete a dashboard.
DeleteMetricStream	<code>cloudwatch:DeleteMetricStream</code> Required to delete a metric stream.
DescribeAlarmHistory	<code>cloudwatch:DescribeAlarmHistory</code> Required to view alarm history. To retrieve information about composite alarms, your <code>cloudwatch:DescribeAlarmHistory</code> permission must have a * scope. You can't return information about composite alarms if your <code>cloudwatch:DescribeAlarmHistory</code> permission has a narrower scope.
DescribeAlarms	<code>cloudwatch:DescribeAlarms</code> Required to retrieve information about alarms. To retrieve information about composite alarms, your <code>cloudwatch:DescribeAlarms</code> permission must have a * scope. You can't return information about composite alarms if your <code>cloudwatch:DescribeAlarms</code> permission has a narrower scope.
DescribeAlarmsForMetric	<code>cloudwatch:DescribeAlarmsForMetric</code> Required to view alarms for a metric.
DisableAlarmActions	<code>cloudwatch:DisableAlarmActions</code> Required to disable an alarm action.
EnableAlarmActions	<code>cloudwatch:EnableAlarmActions</code> Required to enable an alarm action.
GetDashboard	<code>cloudwatch:GetDashboard</code> Required to display data about existing dashboards.
GetMetricData	<code>cloudwatch:GetMetricData</code> Required to graph metric data in the CloudWatch console, to retrieve large batches of metric data, and perform metric math on that data.

CloudWatch API operations	Required permissions (API actions)
GetMetricStatistics	<code>cloudwatch:GetMetricStatistics</code> Required to view graphs in other parts of the CloudWatch console and in dashboard widgets.
GetMetricStream	<code>cloudwatch:GetMetricStream</code> Required to view information about a metric stream.
GetMetricWidgetImage	<code>cloudwatch:GetMetricWidgetImage</code> Required to retrieve a snapshot graph of one or more CloudWatch metrics as a bitmap image.
ListDashboards	<code>cloudwatch>ListDashboards</code> Required to view the list of CloudWatch dashboards in your account.
ListMetrics	<code>cloudwatch>ListMetrics</code> Required to view or search metric names within the CloudWatch console and in the CLI. Required to select metrics on dashboard widgets.
ListMetricStreams	<code>cloudwatch>ListMetricStreams</code> Required to view or search the list of metric streams in the account.
PutCompositeAlarm	<code>cloudwatch:PutCompositeAlarm</code> Required to create a composite alarm. To create a composite alarm, your <code>cloudwatch:PutCompositeAlarm</code> permission must have a * scope. You can't return information about composite alarms if your <code>cloudwatch:PutCompositeAlarm</code> permission has a narrower scope.
PutDashboard	<code>cloudwatch:PutDashboard</code> Required to create a dashboard or update an existing dashboard.
PutMetricAlarm	<code>cloudwatch:PutMetricAlarm</code> Required to create or update an alarm.
PutMetricData	<code>cloudwatch:PutMetricData</code> Required to create metrics.
PutMetricStream	<code>cloudwatch:PutMetricStream</code> Required to create a metric stream.

CloudWatch API operations	Required permissions (API actions)
SetAlarmState	<code>cloudwatch:SetAlarmState</code> Required to manually set an alarm's state.
StartMetricStreams	<code>cloudwatch:StartMetricStreams</code> Required to start the flow of metrics in a metric stream.
StopMetricStreams	<code>cloudwatch:StopMetricStreams</code> Required to temporarily stop the flow of metrics in a metric stream.
TagResource	<code>cloudwatch:TagResource</code> Required to add or update tags on CloudWatch resources such as alarms and Contributor Insights rules.
UntagResource	<code>cloudwatch:UntagResource</code> Required to remove tags from CloudWatch resources .

CloudWatch Contributor Insights API operations and required permissions for actions

Important

When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see [Using condition keys to limit Contributor Insights users' access to log groups \(p. 920\)](#).

CloudWatch Contributor Insights API operations	Required permissions (API actions)
DeleteInsightRules	<code>cloudwatch>DeleteInsightRules</code> Required to delete Contributor Insights rules.
DescribeInsightRules	<code>cloudwatch:DescribeInsightRules</code> Required to view the Contributor Insights rules in your account.
EnableInsightRules	<code>cloudwatch:EnableInsightRules</code> Required to enable Contributor Insights rules.
GetInsightRuleReport	<code>cloudwatch:GetInsightRuleReport</code> Required to retrieve time series data and other statistics collected by Contributor Insights rules.
PutInsightRule	<code>cloudwatch:PutInsightRule</code>

CloudWatch Contributor Insights API operations	Required permissions (API actions)
	Required to create Contributor Insights rules. See the Important note at the beginning of this table.

CloudWatch Events API operations and required permissions for actions

CloudWatch Events API operations	Required permissions (API actions)
DeleteRule	<code>events:DeleteRule</code> Required to delete a rule.
DescribeRule	<code>events:DescribeRule</code> Required to list the details about a rule.
DisableRule	<code>events:DisableRule</code> Required to disable a rule.
EnableRule	<code>events:EnableRule</code> Required to enable a rule.
ListRuleNamesByTarget	<code>events>ListRuleNamesByTarget</code> Required to list rules associated with a target.
ListRules	<code>events>ListRules</code> Required to list all rules in your account.
ListTargetsByRule	<code>events>ListTargetsByRule</code> Required to list all targets associated with a rule.
PutEvents	<code>events:PutEvents</code> Required to add custom events that can be matched to rules.
PutRule	<code>events:PutRule</code> Required to create or update a rule.
PutTargets	<code>events:PutTargets</code> Required to add targets to a rule.
RemoveTargets	<code>events:RemoveTargets</code> Required to remove a target from a rule.
TestEventPattern	<code>events:TestEventPattern</code> Required to test an event pattern against a given event.

CloudWatch Logs API operations and required permissions for actions

CloudWatch Logs API operations	Required permissions (API actions)
CancelExportTask	<code>logs:CancelExportTask</code> Required to cancel a pending or running export task.
CreateExportTask	<code>logs>CreateExportTask</code> Required to export data from a log group to an Amazon S3 bucket.
CreateLogGroup	<code>logs>CreateLogGroup</code> Required to create a new log group.
CreateLogStream	<code>logs>CreateLogStream</code> Required to create a new log stream in a log group.
DeleteDestination	<code>logs>DeleteDestination</code> Required to delete a log destination and disables any subscription filters to it.
DeleteLogGroup	<code>logs>DeleteLogGroup</code> Required to delete a log group and any associated archived log events.
DeleteLogStream	<code>logs>DeleteLogStream</code> Required to delete a log stream and any associated archived log events.
DeleteMetricFilter	<code>logs>DeleteMetricFilter</code> Required to delete a metric filter associated with a log group.
DeleteQueryDefinition	<code>logs>DeleteQueryDefinition</code> Required to delete a saved query definition in CloudWatch Logs Insights.
DeleteResourcePolicy	<code>logs>DeleteResourcePolicy</code> Required to delete a CloudWatch Logs resource policy.
DeleteRetentionPolicy	<code>logs>DeleteRetentionPolicy</code> Required to delete a log group's retention policy.
DeleteSubscriptionFilter	<code>logs>DeleteSubscriptionFilter</code>

CloudWatch Logs API operations	Required permissions (API actions)
	Required to delete the subscription filter associated with a log group.
DescribeDestinations	<code>logs:DescribeDestinations</code> Required to view all destinations associated with the account.
DescribeExportTasks	<code>logs:DescribeExportTasks</code> Required to view all export tasks associated with the account.
DescribeLogGroups	<code>logs:DescribeLogGroups</code> Required to view all log groups associated with the account.
DescribeLogStreams	<code>logs:DescribeLogStreams</code> Required to view all log streams associated with a log group.
DescribeMetricFilters	<code>logs:DescribeMetricFilters</code> Required to view all metrics associated with a log group.
DescribeQueryDefinitions	<code>logs:DescribeQueryDefinitions</code> Required to see the list of saved query definitions in CloudWatch Logs Insights.
DescribeQueries	<code>logs:DescribeQueries</code> Required to see the list of CloudWatch Logs Insights queries that are scheduled, executing, or have recently executed.
DescribeResourcePolicies	<code>logs:DescribeResourcePolicies</code> Required to view a list of CloudWatch Logs resource policies.
DescribeSubscriptionFilters	<code>logs:DescribeSubscriptionFilters</code> Required to view all subscription filters associated with a log group.
FilterLogEvents	<code>logs:FilterLogEvents</code> Required to sort log events by log group filter pattern.
GetLogEvents	<code>logs:GetLogEvents</code> Required to retrieve log events from a log stream.

CloudWatch Logs API operations	Required permissions (API actions)
GetLogGroupFields	<code>logs:GetLogGroupFields</code> Required to retrieve the list of fields that are included in the log events in a log group.
GetLogRecord	<code>logs:GetLogRecord</code> Required to retrieve the details from a single log event.
GetQueryResults	<code>logs:GetQueryResults</code> Required to retrieve the results of CloudWatch Logs Insights queries.
ListTagsLogGroup	<code>logs>ListTagsLogGroup</code> Required to list the tags associated with a log group.
PutDestination	<code>logs:PutDestination</code> Required to create or update a destination log stream (such as an Kinesis stream).
PutDestinationPolicy	<code>logs:PutDestinationPolicy</code> Required to create or update an access policy associated with an existing log destination.
PutLogEvents	<code>logs:PutLogEvents</code> Required to upload a batch of log events to a log stream.
PutMetricFilter	<code>logs:PutMetricFilter</code> Required to create or update a metric filter and associate it with a log group.
PutQueryDefinition	<code>logs:PutQueryDefinition</code> Required to save a query in CloudWatch Logs Insights.
PutResourcePolicy	<code>logs:PutResourcePolicy</code> Required to create a CloudWatch Logs resource policy.
PutRetentionPolicy	<code>logs:PutRetentionPolicy</code> Required to set the number of days to keep log events (retention) in a log group.
PutSubscriptionFilter	<code>logs:PutSubscriptionFilter</code> Required to create or update a subscription filter and associate it with a log group.

CloudWatch Logs API operations	Required permissions (API actions)
StartQuery	<code>logs:StartQuery</code> Required to start CloudWatch Logs Insights queries.
StopQuery	<code>logs:StopQuery</code> Required to stop a CloudWatch Logs Insights query that is in progress.
TagLogGroup	<code>logs:TagLogGroup</code> Required to add or update log group tags.
TestMetricFilter	<code>logs:TestMetricFilter</code> Required to test a filter pattern against a sampling of log event messages.

Amazon EC2 API operations and required permissions for actions

Amazon EC2 API operations	Required permissions (API actions)
DescribeInstanceStatus	<code>ec2:DescribeInstanceStatus</code> Required to view EC2 instance status details.
DescribeInstances	<code>ec2:DescribeInstances</code> Required to view EC2 instance details.
RebootInstances	<code>ec2:RebootInstances</code> Required to reboot an EC2 instance.
StopInstances	<code>ec2:StopInstances</code> Required to stop an EC2 instance.
TerminateInstances	<code>ec2:TerminateInstances</code> Required to terminate an EC2 instance.

Amazon EC2 Auto Scaling API operations and required permissions for actions

Amazon EC2 Auto Scaling API operations	Required permissions (API actions)
Scaling	<code>autoscaling:Scaling</code> Required to scale an Auto Scaling group.

Amazon EC2 Auto Scaling API operations	Required permissions (API actions)
Trigger	<code>autoscaling:Trigger</code> Required to trigger an Auto Scaling action.

Compliance validation for Amazon CloudWatch

Third-party auditors assess the security and compliance of Amazon CloudWatch 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 Amazon CloudWatch is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [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.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – 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.

Resilience in Amazon CloudWatch

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

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Infrastructure security in Amazon CloudWatch

As a managed service, Amazon CloudWatch is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Amazon CloudWatch through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also

support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Network isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the Amazon Web Services Cloud. A subnet is a range of IP addresses in a VPC. You can deploy a variety of AWS resources in the subnets of your VPCs. For example, you can deploy Amazon EC2 instances, EMR clusters, and DynamoDB tables in subnets. For more information, see the [Amazon VPC User Guide](#).

To enable CloudWatch to communicate with resources in a VPC without going through the public internet, use AWS PrivateLink. For more information, see [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints \(p. 952\)](#).

A private subnet is a subnet with no default route to the public internet. Deploying an AWS resource in a private subnet does not prevent Amazon CloudWatch from collecting built-in metrics from the resource.

If you need to publish custom metrics from an AWS resource in a private subnet, you can do so using a proxy server. The proxy server forwards those HTTPS requests to the public API endpoints for CloudWatch.

Using CloudWatch and CloudWatch Synthetics with interface 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, CloudWatch, and CloudWatch Synthetics. You can use these connections to enable CloudWatch and CloudWatch Synthetics to communicate with your resources on your VPC without going through 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. To connect your VPC to CloudWatch or CloudWatch Synthetics, you define an *interface VPC endpoint* to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to CloudWatch or CloudWatch Synthetics without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see [What Is Amazon VPC](#) in the [Amazon VPC User Guide](#).

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see the [New – AWS PrivateLink for AWS Services](#) blog post.

The following steps are for users of Amazon VPC. For more information, see [Getting Started](#) in the [Amazon VPC User Guide](#).

CloudWatch VPC endpoint

CloudWatch currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)

- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Creating a VPC endpoint for CloudWatch

To start using CloudWatch with your VPC, create an interface VPC endpoint for CloudWatch. The service name to choose is `com.amazonaws.region.monitoring`. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch. CloudWatch calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch, and you already have metrics flowing to CloudWatch from resources located on your VPC, these metrics begin flowing through the interface VPC endpoint by default.

Controlling access to your CloudWatch VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch. This policy allows users connecting to CloudWatch through the VPC to send metric data to CloudWatch and prevents them from performing other CloudWatch actions.

```
{  
  "Statement": [  
    {  
      "Sid": "PutOnly",
```

```
    "Principal": "*",
    "Action": [
        "cloudwatch:PutMetricData"
    ],
    "Effect": "Allow",
    "Resource": "*"
}
]
```

To edit the VPC endpoint policy for CloudWatch

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch, choose **Create Endpoint**. Select **com.amazonaws.region.monitoring**, and then choose **Create endpoint**.
4. Select the **com.amazonaws.region.monitoring** endpoint, and then choose the **Policy** tab.
5. Choose **Edit Policy**, and then make your changes.

CloudWatch Synthetics VPC endpoint

CloudWatch Synthetics currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)

Creating a VPC endpoint for CloudWatch Synthetics

To start using CloudWatch Synthetics with your VPC, create an interface VPC endpoint for CloudWatch Synthetics. The service name to choose is **com.amazonaws.region.synthetics**. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch Synthetics. CloudWatch Synthetics communicates with other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch

Synthetics, and you already have an interface endpoint for Amazon S3, CloudWatch Synthetics begins communicating with Amazon S3 through the interface VPC endpoint by default.

Controlling access to your CloudWatch Synthetics VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, we attach a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies affect canaries that are managed privately by VPC. They are not needed for canaries that run on private subnets.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch Synthetics. This policy enables users connecting to CloudWatch Synthetics through the VPC to view information about canaries and their runs, but not to create, modify, or delete canaries.

```
{  
    "Statement": [  
        {  
            "Action": [  
                "synthetics:DescribeCanaries",  
                "synthetics:GetCanaryRuns"  
            ],  
            "Effect": "Allow",  
            "Resource": "*",  
            "Principal": "*"  
        }  
    ]  
}
```

To edit the VPC endpoint policy for CloudWatch Synthetics

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch Synthetics, choose **Create Endpoint**. Select **com.amazonaws.*region*.synthetics** and then choose **Create endpoint**.
4. Select the **com.amazonaws.*region*.synthetics** endpoint and then choose the **Policy** tab.
5. Choose **Edit Policy**, and then make your changes.

Security considerations for Synthetics canaries

The following sections explain security issues that you should consider when creating and running canaries in Synthetics.

Use secure connections

Because canary code and the results from canary test runs can contain sensitive information, do not have your canary connect to endpoints over unencrypted connections. Always use encrypted connections, such as those that begin with `https://`.

Canary naming considerations

The Amazon Resource Name (ARN) of a canary is included in the user-agent header as a part of outbound calls made from the Puppeteer-driven Chromium browser that is included as a part of the CloudWatch Synthetics wrapper library. This helps identify CloudWatch Synthetics canary traffic and relate it back to the canaries that are making calls.

The canary ARN includes the canary name. Choose canary names that do not reveal proprietary information.

Additionally, be sure to point your canaries only at websites and endpoints that you control.

Secrets in canary code

We recommend that you don't include secrets, such as access keys or database credentials, in your canary source code. For more information about how to use AWS Secrets Manager to help keep your secrets safe, see [What is AWS Secrets Manager?](#).

Permissions considerations

We recommend that you restrict access to resources that are created or used by CloudWatch Synthetics. Use tight permissions on the Amazon S3 buckets where canaries store test run results and other artifacts, such as logs and screenshots.

Similarly, keep tight permissions on the locations where your canary source code is stored, so that no user accidentally or maliciously deletes the Lambda layers or Lambda functions used for the canary.

To help make sure you run the canary code you intend, you can use object versioning on the Amazon S3 bucket where your canary code is stored. Then when you specify this code to run as a canary, you can include the object `versionId` as part of the path, as in the following examples.

```
https://bucket.s3.amazonaws.com/path/object.zip?versionId=version-id
https://s3.amazonaws.com/bucket/path/object.zip?versionId=version-id
https://bucket.s3-region.amazonaws.com/path/object.zip?versionId=version-id
```

Stack traces and exception messages

By default, CloudWatch Synthetics canaries capture any exception thrown by your canary script, no matter whether the script is custom or is from a blueprint. CloudWatch Synthetics logs both the exception message and the stack trace to three locations:

- Back into the CloudWatch Synthetics service to speed up debugging when you describe test runs
- Into CloudWatch Logs according to the configuration that your Lambda functions are created with
- Into the Synthetics log file, which is a plaintext file that is uploaded to the Amazon S3 location specified by the value you set for the `resultsLocation` of the canary

If you want to send and store less information, you can capture exceptions before they return to the CloudWatch Synthetics wrapper library.

You can also have request URLs in your errors. CloudWatch Synthetics scans for any URLs in the error thrown by your script and redacts restricted URL parameters from them based on the `restrictedUrlParameters` configuration. If you are logging error messages in your script, you can use [???](#) (p. 212) to redact URLs before logging.

Scope your IAM roles narrowly

We recommend that you do not configure your canary to visit potentially malicious URLs or endpoints. Pointing your Canary to untrusted or unknown websites or endpoints could expose your Lambda function code to malicious user's scripts. Assuming a malicious website can break out of Chromium, it could have access to your Lambda code in a similar way to if you connected to it using an internet browser.

Run your Lambda function with an IAM execution role that has scoped-down permissions. This way, if your Lambda function is compromised by a malicious script, it is limited in the actions it can take when running as your canary's AWS account.

When you use the CloudWatch console to create a canary, it is created with a scoped-down IAM execution role.

Sensitive data redaction

CloudWatch Synthetics captures URLs, status code, failure reason (if any), and headers and bodies of requests and responses. This enables a canary user to understand, monitor, and debug canaries.

The configurations described in the following sections can be set at any point in canary execution. You can also choose to apply different configurations to different synthetics steps.

Request URLs

By default, CloudWatch Synthetics logs request URLs, status codes, and the status reason for each URL in canary logs. Request URLs can also appear in canary execution reports, HAR files, and so on. Your request URL might contain sensitive query parameters, such as access tokens or passwords. You can redact sensitive information from being logged by CloudWatch Synthetics.

To redact sensitive information, set the configuration property **restrictedUrlParameters**. For more information, see [SyntheticsConfiguration class \(p. 201\)](#). This causes CloudWatch Synthetics to redact URL parameters, including path and query parameter values, based on **restrictedUrlParameters** before logging. If you are logging URLs in your script, you can use [???](#) (p. 211) to redact URLs before logging. For more information, see [SyntheticsLogHelper class \(p. 210\)](#).

Headers

By default, CloudWatch Synthetics doesn't log request/response headers. For UI canaries, this is the default behavior for canaries using runtime version `syn-nodejs-puppeteer-3.2` and later.

If your headers don't contain sensitive information, you can enable headers in HAR file and HTTP reports by setting the **includeRequestHeaders** and **includeResponseHeaders** properties to `true`. You can enable all headers but choose to restrict values of sensitive header keys. For example, you can choose to only redact Authorization headers from artifacts produced by canaries.

Request and response body

By default, CloudWatch Synthetics doesn't log the request/response body in canary logs or reports. This information is particularly useful for API canaries. Synthetics captures all HTTP requests and can show headers, request and response bodies. For more information, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\) \(p. 220\)](#). You can choose to enable request/response body by setting the **includeRequestBody** and **includeResponseBody** properties to `true`.

Logging Amazon CloudWatch API calls with AWS CloudTrail

Amazon CloudWatch and CloudWatch Synthetics are integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls made by or on behalf of your AWS account. The captured calls include calls from the console and code calls to API operations.

If you create a *trail*, you can enable continuous delivery of CloudTrail events to an S3 bucket, including events for CloudWatch. 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 CloudWatch, the IP address from which the request was made, who made the request, when it was made, and other details.

To learn more about CloudTrail, including how to configure and enable it, see 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 AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

For an ongoing record of events in your AWS account, including events for CloudWatch and CloudWatch Synthetics, create a trail. A trail enables CloudTrail to deliver log files to an 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](#)

Topics

- [CloudWatch information in CloudTrail \(p. 958\)](#)
- [CloudWatch Synthetics information in CloudTrail \(p. 961\)](#)

CloudWatch information in CloudTrail

CloudWatch supports logging the following actions as events in CloudTrail log files:

- [DeleteAlarms](#)
- [DeleteAnomalyDetector](#)
- [DeleteDashboards](#)
- [DescribeAlarmHistory](#)
- [DescribeAlarms](#)
- [DescribeAlarmsForMetric](#)
- [DescribeAnomalyDetectors](#)
- [DisableAlarmActions](#)
- [EnableAlarmActions](#)
- [GetDashboard](#)
- [ListDashboards](#)
- [PutAnomalyDetector](#)
- [PutDashboard](#)
- [PutMetricAlarm](#)
- [SetAlarmState](#)

Example: CloudWatch log file entries

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

```
{
    "Records": [
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "Root",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:root",
                "accountId": "123456789012",
                "accessKeyId": "EXAMPLE_KEY_ID"
            },
            "eventTime": "2014-03-23T21:50:34Z",
            "eventSource": "monitoring.amazonaws.com",
            "eventName": "PutMetricAlarm",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
            "requestParameters": {
                "threshold": 50.0,
                "period": 60,
                "metricName": "CloudTrail Test",
                "evaluationPeriods": 3,
                "comparisonOperator": "GreaterThanOrEqualToThreshold",
                "namespace": "AWS/CloudWatch",
                "alarmName": "CloudTrail Test Alarm",
                "statistic": "Sum"
            },
            "responseElements": null,
            "requestID": "29184022-b2d5-11e3-a63d-9b463e6d0ff0",
            "eventID": "b096d5b7-dcf2-4399-998b-5a53eca76a27"
        },
        ..additional entries
    ]
}
```

The following log file entry shows that a user called the CloudWatch Events `PutRule` action.

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2015-11-17T23:56:15Z"
            }
        }
    },
    "eventTime": "2015-11-18T00:11:28Z",
    "eventSource": "events.amazonaws.com",
    "eventName": "PutRule",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS CloudWatch Console",
    "requestParameters": {
        "description": "",
        "name": "cttest2",
        "state": "ENABLED",
        "eventPattern": "{\"source\":[\"aws.ec2\"], \"detail-type\":[\"EC2 Instance State-change Notification\"]}",
        "scheduleExpression": ""
    },
    "responseElements": {
        "ruleArn": "arn:aws:events:us-east-1:123456789012:rule/cttest2"
    },
    "requestID": "e9caf887-8d88-11e5-a331-3332aa445952",
    "eventID": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
    "eventType": "AwsApiCall",
    "apiVersion": "2015-10-07",
    "recipientAccountId": "123456789012"
}
```

The following log file entry shows that a user called the CloudWatch Logs `CreateExportTask` action.

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/someuser",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "someuser"
    },
    "eventTime": "2016-02-08T06:35:14Z",
    "eventSource": "logs.amazonaws.com",
    "eventName": "CreateExportTask",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
    "requestParameters": {
        "destination": "yourdestination",
        "logGroupName": "yourloggroup",
        "to": 123456789012,
        "from": 0,
        "taskName": "yourtask"
    },
}
```

```
    "responseElements": {
        "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
    },
    "requestID": "1cd74c1c-ce2e-12e6-99a9-8ddb26bd06c9",
    "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
    "eventType": "AwsApiCall",
    "apiVersion": "20140328",
    "recipientAccountId": "123456789012"
}
```

CloudWatch Synthetics information in CloudTrail

CloudWatch Synthetics supports logging the following actions as events in CloudTrail log files:

- [CreateCanary](#)
- [DeleteCanary](#)
- [DescribeCanaries](#)
- [DescribeCanariesLastRun](#)
- [DescribeRuntimeVersions](#)
- [GetCanary](#)
- [GetCanaryRuns](#)
- [ListTagsForResource](#)
- [StartCanary](#)
- [StopCanary](#)
- [TagResource](#)
- [UntagResource](#)
- [UpdateCanary](#)

Example: CloudWatch Synthetics log file entries

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

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    }
}
```

```

        },
        "eventTime": "2020-04-08T23:06:47Z",
        "eventSource": "synthetics.amazonaws.com",
        "eventName": "DescribeCanaries",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
java/1.8.0_212 vendor/Oracle_Corporation",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "201ed5f3-15db-4f87-94a4-123456789",
        "eventId": "73ddbd81-3dd0-4ada-b246-123456789",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "111122223333"
    }
}

```

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

```

{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "UpdateCanary",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Schedule": {
            "Expression": "rate(1 minute)"
        },
        "name": "sample_canary_name",
        "Code": {
            "Handler": "myOwnScript.handler",
            "ZipFile": "SAMPLE_ZIP_FILE"
        }
    },
    "responseElements": null,
}

```

```
    "requestID": "fe4759b0-0849-4e0e-be71-1234567890",
    "eventID": "9dc60c83-c3c8-4fa5-bd02-1234567890",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

The following example shows a CloudTrail Synthetics log entry that demonstrates the GetCanaryRuns action.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "GetCanaryRuns",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Filter": "TIME_RANGE",
        "name": "sample_canary_name",
        "FilterValues": [
            "2020-04-08T23:00:00.000Z",
            "2020-04-08T23:10:00.000Z"
        ]
    },
    "responseElements": null,
    "requestID": "2f56318c-cfb9-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Grafana integration

You can use Grafana version 6.5.0 and later to contextually advance through the CloudWatch console and query a dynamic list of metrics by using wildcards. This can help you monitor metrics for AWS resources, such as Amazon Elastic Compute Cloud instances or containers. When new instances are created as part of an Auto Scaling event, they appear in the graph automatically. You don't need to track the new instance IDs. Prebuilt dashboards help simplify the getting started experience for monitoring Amazon EC2, Amazon Elastic Block Store, and AWS Lambda resources.

You can use Grafana version 7.0 and later to perform CloudWatch Logs Insights queries on log groups in CloudWatch Logs. You can visualize your query results in bar, line, and stacked graphs and in a table format. For more information about CloudWatch Logs Insights, see [Analyzing Log Data with CloudWatch Logs Insights](#).

For more information about how to get started, see [Using AWS CloudWatch in Grafana](#) in the Grafana Labs documentation.

CloudWatch service quotas

CloudWatch has the following quotas for metrics, alarms, API requests, and alarm email notifications.

Resource	Default quota
Alarm actions	5/alarm. This quota cannot be changed.
Alarms	10/month/customer for free. Additional alarms incur charges. No limit on the total number of alarms per account. Alarms based on metric math expressions can have up to 10 metrics.
Anomaly detection models	500 per Region, per account.
API requests	1,000,000/month/customer for free.
Canaries	200 per Region per account. You can request a quota increase .
Contributor Insights API requests	The following APIs have a quota of 20 transactions per second (TPS) and per Region. <ul style="list-style-type: none"> • DescribeInsightRules The quota cannot be changed. • GetInsightRuleReport You can request a quota increase. The following APIs have a quota of 5 TPS per Region. This quota cannot be changed. <ul style="list-style-type: none"> • DeleteInsightRules • PutInsightRule The following APIs have a quota of 1 TPS per Region. This quota cannot be changed. <ul style="list-style-type: none"> • DisableInsightRules • EnableInsightRules
Contributor Insights rules	100 rules per Region per account. You can request a quota increase .
Custom metrics	No quota.
Dashboards	Up to 500 metrics per dashboard widget. Up to 2500 metrics per dashboard, across all widgets.

Resource	Default quota
	<p>These quotas include all metrics retrieved for use in metric math functions, even if those metrics are not displayed on the graph.</p> <p>These quotas cannot be changed.</p>
DescribeAlarms	<p>9 transactions per second (TPS) per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
DeleteAlarms request DescribeAlarmHistory request DisableAlarmActions request EnableAlarmActions request SetAlarmState request	<p>3 TPS per Region for each of these operations. The maximum number of operation requests you can make per second without being throttled.</p> <p>These quotas cannot be changed.</p>
DescribeAlarmsForMetric request	<p>9 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>This quotas cannot be changed.</p>
DeleteDashboards request GetDashboard request ListDashboards request PutDashboard request	<p>10 TPS per Region for each of these operations. The maximum number of operation requests you can make per second without being throttled.</p> <p>These quotas cannot be changed.</p>
PutAnomalyDetector DescribeAnomalyDetectors	<p>10 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p>
DeleteAnomalyDetector	<p>5 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p>
Dimensions	<p>10/metric. This quota cannot be changed.</p>

Resource	Default quota
GetMetricData	<p>10 TPS per Region for operations that include Metrics Insights queries. For operations that do not include Metrics Insights queries, the quota is 50 TPS per Region. This is the maximum number of operation requests you can make per second without being throttled. You can request a quota increase.</p> <p>For GetMetricData operations that include a Metrics Insights query, the quota is 4,300,000 Datapoints Per Second (DPS) for the most recent 3 hours. This is calculated against the total number of data points scanned by the query (which can include no more than 10,000 metrics.)</p> <p>180,000 Datapoints Per Second (DPS) if the <code>StartTime</code> used in the API request is less than or equal to three hours from current time. 396,000 DPS if the <code>StartTime</code> is more than three hours from current time. This is the maximum number of datapoints you can request per second using one or more API calls without being throttled. This quota cannot be changed.</p> <p>The DPS is calculated based on estimated data points, not actual data points. The data point estimate is calculated using the requested time range, period, and retention period. This means that if the actual data points in the requested metrics are sparse or empty, throttling still occurs if the estimated data points exceed the quota. The DPS quota is per-Region.</p>
GetMetricData	<p>A single GetMetricData call can include as many as 500 <code>MetricDataQuery</code> structures.</p> <p>This quota cannot be changed.</p>
GetMetricStatistics	<p>400 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
GetMetricWidgetImage	<p>Up to 500 metrics per image. This quota cannot be changed.</p> <p>20 TPS per Region. The maximum number of operation requests you can make per second without being throttled. This quota cannot be changed.</p>
ListMetrics	<p>25 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
Metric data storage	15 months. This quota cannot be changed.
Metric data values	The value of a metric data point must be within the range of -2^360 to 2^360. Special values (for example, NaN, +Infinity, -Infinity) are not supported. This quota cannot be changed.

Resource	Default quota
MetricDatum items	20/ PutMetricData request. A MetricDatum object can contain a single value or a StatisticSet object representing many values. This quota cannot be changed.
Metrics	10/month/customer for free.
Metrics Insights queries	A single query can process no more than 10,000 metrics. This means that if the SELECT , FROM , and WHERE clauses would match more than 10,000 metrics, only the first 10,000 of these metrics that are found will be processed by the query. A single query can return no more than 500 time series.
Alarm evaluation period	The maximum value, calculated by multiplying the alarm period by the number of evaluation periods used, is one day (86,400 seconds). This quota cannot be changed.
PutMetricAlarm request	3 TPS per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase .
PutMetricData request	40 KB for HTTP POST requests. PutMetricData can handle 150 transactions per second (TPS), which is the maximum number of operation requests you can make per second without being throttled. You can request a quota increase .
Amazon SNS email notifications	1,000/month/customer for free.

Document history

The following table describes important changes in each release of the *Amazon CloudWatch User Guide*, beginning in June 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
Amazon CloudWatch Application Insights support for containerized applications and microservices from the Container Insights console. (p. 603)	You can display CloudWatch Application Insights detected problems for Amazon ECS and Amazon EKS on your Container Insights dashboard.	November 17, 2021
Amazon CloudWatch Application Insights monitoring for SAP HANA databases. (p. 603)	You can monitor SAP HANA databases with Application Insights.	November 15, 2021
Amazon CloudWatch Application Insights support for monitoring all resources in an account. (p. 603)	You can onboard and monitor all resources in an account.	September 15, 2021
Amazon CloudWatch Application Insights support for Amazon FSx. (p. 603)	You can monitor metrics retrieved from Amazon FSx.	August 31, 2021
SDK Metrics is no longer supported. (p. 969)	CloudWatch SDK Metrics is no longer supported.	August 25, 2021
Amazon CloudWatch Application Insights support for setting up container monitoring. (p. 603)	You can monitor containers using best practices with Amazon CloudWatch Application Insights.	May 18, 2021
Metric streams is generally available (p. 969)	You can use metric streams to continually stream CloudWatch metrics to a destination of your choice. For more information, see Metric streams in the <i>Amazon CloudWatch User Guide</i> .	March 31, 2021
Amazon CloudWatch Application Insights monitoring for Oracle databases on Amazon RDS and Amazon EC2. (p. 603)	You can monitor metrics and logs retrieved from Oracle with Amazon CloudWatch Application Insights.	January 16, 2021
Lambda Insights is generally available (p. 969)	CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. For more information, see Using Lambda Insights in the <i>Amazon CloudWatch User Guide</i> .	December 3, 2020

Amazon CloudWatch Application Insights monitoring for Prometheus JMX exporter metrics. (p. 603)	You can monitor metrics retrieved from Prometheus JMX exporter with Amazon CloudWatch Application Insights.	November 20, 2020
CloudWatch Synthetics releases new runtime version (p. 969)	CloudWatch Synthetics has released a new runtime version. For more information, see Canary Runtime Versions in the <i>Amazon CloudWatch User Guide</i> .	September 11, 2020
Amazon CloudWatch Application Insights monitoring for PostgreSQL on Amazon RDS and Amazon EC2. (p. 603)	You can monitor applications built with PostgreSQL running on Amazon RDS or Amazon EC2.	September 11, 2020
CloudWatch supports dashboard sharing (p. 969)	You can now share CloudWatch dashboards with people outside of your organization and AWS account. For more information, see Sharing CloudWatch Dashboards in the <i>Amazon CloudWatch User Guide</i> .	September 10, 2020
Set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights (p. 603)	You can use the documentation tutorial to help you to set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights.	August 19, 2020
AWS CloudFormation support for Amazon CloudWatch Application Insights applications. (p. 603)	You can add CloudWatch Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from AWS CloudFormation templates.	July 30, 2020
Amazon CloudWatch Application Insights monitoring for Aurora for MySQL database clusters. (p. 603)	You can monitor Aurora for MySQL database clusters (RDS Aurora) with Amazon CloudWatch Application Insights.	July 2, 2020
CloudWatch Contributor Insights general availability (p. 969)	CloudWatch Contributor Insights is now generally available. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see Using Contributor Insights to Analyze High-Cardinality Data in the <i>Amazon CloudWatch User Guide</i> .	April 2, 2020

CloudWatch Synthetics public preview (p. 969)	CloudWatch Synthetics is now in public preview. It enables you to create canaries to monitor your endpoints and APIs. For more information, see Using Canaries in the <i>Amazon CloudWatch User Guide</i> .	November 25, 2019
CloudWatch Contributor Insights public preview (p. 969)	CloudWatch Contributor Insights is now in public preview. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see Using Contributor Insights to Analyze High-Cardinality Data in the <i>Amazon CloudWatch User Guide</i> .	November 25, 2019
CloudWatch launches ServiceLens feature (p. 969)	ServiceLens enhances the observability of your services and applications by enabling you to integrate traces, metrics, logs, and alarms into one place. ServiceLens integrates CloudWatch with AWS X-Ray to provide an end-to-end view of your application. For more information, see Using ServiceLens to Monitor the Health of Your Applications in the <i>Amazon CloudWatch User Guide</i> .	November 21, 2019
Use CloudWatch to proactively manage your AWS service quotas (p. 969)	You can use CloudWatch to proactively manage your AWS service quotas. CloudWatch usage metrics provide visibility into your account's usage of resources and API operations. For more information, see Service Quotas Integration and Usage Metrics in the <i>Amazon CloudWatch User Guide</i> .	November 19, 2019
CloudWatch sends events when alarms change state (p. 969)	CloudWatch now sends an event to Amazon EventBridge when any CloudWatch alarm changes state. For more information, see Alarm Events and EventBridge in the <i>Amazon CloudWatch User Guide</i> .	October 8, 2019

Container Insights (p. 969)	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Using Container Insights in the Amazon CloudWatch User Guide .	August 30, 2019
Updates for Container Insights preview metrics on Amazon EKS and Kubernetes (p. 969)	The Container Insights on Amazon EKS and Kubernetes public preview has been updated. Instanceld is now included as a dimension to the cluster EC2 instances. This allows alarms that have been created on these metrics to trigger the following EC2 actions: Stop, Terminate, Reboot, or Recover. Additionally, pod and service metrics are now reported by Kubernetes namespace to simplify the monitoring and alarming on metrics by namespace.	August 19, 2019
Updates for AWS Systems Manager OpsCenter integration (p. 969)	Updates on how CloudWatch Application Insights integrates with Systems Manager OpsCenter.	August 7, 2019
CloudWatch usage metrics (p. 969)	CloudWatch usage metrics help you track the usage of your CloudWatch resources and stay within your service limits. For more information, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Usage-Metrics.html .	August 6, 2019
CloudWatch Container Insights public preview (p. 969)	CloudWatch Container Insights is now in public preview. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Using Container Insights in the Amazon CloudWatch User Guide .	July 9, 2019

CloudWatch Anomaly Detection public preview (p. 969)	CloudWatch anomaly detection is now in public preview. CloudWatch applies machine-learning algorithms to a metric's past data to create a model of the metric's expected values. You can use this model for visualization and for setting alarms. For more information, see Using CloudWatch Anomaly Detection in the <i>Amazon CloudWatch User Guide</i> .	July 9, 2019
CloudWatch Application Insights for .NET and SQL Server (p. 969)	CloudWatch Application Insights for .NET and SQL Server facilitates observability for .NET and SQL Server applications. It can help you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications.	June 21, 2019
CloudWatch agent section reorganized (p. 969)	The CloudWatch agent documentation has been rewritten to improve clarity, especially for customers using the command line to install and configure the agent. For more information, see Collecting Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent in the <i>Amazon CloudWatch User Guide</i> .	March 28, 2019
SEARCH function added to metric math expressions (p. 969)	You can now use a SEARCH function in metric math expressions. This enables you to create dashboards that update automatically as new resources are created that match the search query. For more information, see Using Search Expressions in Graphs in the <i>Amazon CloudWatch User Guide</i> .	March 21, 2019
AWS SDK Metrics for Enterprise Support (p. 969)	SDK Metrics helps you assess the health of your AWS services and diagnose latency caused by reaching your account usage limits or by a service outage. For more information, see Monitor Applications Using AWS SDK Metrics in the <i>Amazon CloudWatch User Guide</i> .	December 11, 2018

Alarms on math expressions (p. 969)	CloudWatch supports creating alarms based on metric math expressions. For more information, see Alarms on Math Expressions in the <i>Amazon CloudWatch User Guide</i> .	November 20, 2018
New CloudWatch console home page (p. 969)	Amazon has created a new home page in the CloudWatch console, which automatically displays key metrics and alarms for all the AWS services you are using. For more information, see Getting Started with Amazon CloudWatch in the <i>Amazon CloudWatch User Guide</i> .	November 19, 2018
AWS CloudFormation templates for the CloudWatch Agent (p. 969)	Amazon has uploaded AWS CloudFormation templates that you can use to install and update the CloudWatch agent. For more information, see Install the CloudWatch Agent on New Instances Using AWS CloudFormation in the <i>Amazon CloudWatch User Guide</i> .	November 9, 2018
Enhancements to the CloudWatch Agent (p. 969)	The CloudWatch agent has been updated to work with both the StatsD and collectd protocols. It also has improved cross-account support. For more information, see Retrieve Custom Metrics with StatsD , Retrieve Custom Metrics with collectd , and Sending Metrics and Logs to a Different AWS Account in the <i>Amazon CloudWatch User Guide</i> .	September 28, 2018
Support for Amazon VPC endpoints (p. 969)	You can now establish a private connection between your VPC and CloudWatch. For more information, see Using CloudWatch with Interface VPC Endpoints in the <i>Amazon CloudWatch User Guide</i> .	June 28, 2018

The following table describes important changes to the *Amazon CloudWatch User Guide* before June 2018.

Change	Description	Release date
Metric math	You can now perform math expressions on CloudWatch metrics, producing new time series that you can add to graphs on your dashboard. For more information, see Using metric math (p. 97) .	April 4, 2018

Change	Description	Release date
"M out of N" alarms	You can now configure an alarm to trigger based on "M out of N" datapoints in any alarm evaluation interval. For more information, see Evaluating an alarm (p. 131) .	December 8, 2017
CloudWatch agent	A new unified CloudWatch agent was released. You can use the unified multi-platform agent to collect custom both system metrics and log files from Amazon EC2 instances and on-premises servers. The new agent supports both Windows and Linux and enables customization of metrics collected, including sub-resource metrics such as per-CPU core. For more information, see Collecting metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent (p. 482) .	September 7, 2017
NAT gateway metrics	Added metrics for Amazon VPC NAT gateway.	September 7, 2017
High-resolution metrics	You can now optionally set up custom metrics as high-resolution metrics, with a granularity of as low as one second. For more information, see High-resolution metrics (p. 95) .	July 26, 2017
Dashboard APIs	You can now create, modify, and delete dashboards using APIs and the AWS CLI. For more information, see Creating a CloudWatch dashboard (p. 19) .	July 6, 2017
AWS Direct Connect metrics	Added metrics for AWS Direct Connect.	June 29, 2017
Amazon VPC VPN metrics	Added metrics for Amazon VPC VPN.	May 15, 2017
AppStream 2.0 metrics	Added metrics for AppStream 2.0.	March 8, 2017
CloudWatch console color picker	You can now choose the color for each metric on your dashboard widgets. For more information, see Edit a graph on a CloudWatch dashboard (p. 26) .	February 27, 2017
Alarms on dashboards	Alarms can now be added to dashboards. For more information, see Add an alarm widget to a CloudWatch dashboard (p. 32) .	February 15, 2017
Added metrics for Amazon Polly	Added metrics for Amazon Polly.	December 1, 2016
Added metrics for Amazon Kinesis Data Analytics	Added metrics for Amazon Kinesis Data Analytics.	December 1, 2016
Added support for percentile statistics	You can specify any percentile, using up to two decimal places (for example, p95.45). For more information, see Percentiles (p. 7) .	November 17, 2016

Change	Description	Release date
Added metrics for Amazon Simple Email Service	Added metrics for Amazon Simple Email Service.	November 2, 2016
Updated metrics retention	Amazon CloudWatch now retains metrics data for 15 months instead of 14 days.	November 1, 2016
Updated metrics console interface	The CloudWatch console is updated with improvements to existing functionality and new functionality.	November 1, 2016
Added metrics for Amazon Elastic Transcoder	Added metrics for Amazon Elastic Transcoder.	September 20, 2016
Added metrics for Amazon API Gateway	Added metrics for Amazon API Gateway.	September 9, 2016
Added metrics for AWS Key Management Service	Added metrics for AWS Key Management Service.	September 9, 2016
Added metrics for the new Application Load Balancers supported by Elastic Load Balancing	Added metrics for Application Load Balancers.	August 11, 2016
Added new NetworkPacketsIn and NetworkPacketsOut metrics for Amazon EC2	Added new NetworkPacketsIn and NetworkPacketsOut metrics for Amazon EC2.	March 23, 2016
Added new metrics for Amazon EC2 Spot fleet	Added new metrics for Amazon EC2 Spot fleet.	March 21, 2016
Added new CloudWatch Logs metrics	Added new CloudWatch Logs metrics.	March 10, 2016
Added Amazon OpenSearch Service and AWS WAF metrics and dimensions	Added Amazon OpenSearch Service and AWS WAF metrics and dimensions.	October 14, 2015

Change	Description	Release date
Added support for CloudWatch dashboards	Dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those that are spread out across different Regions. For more information, see Using Amazon CloudWatch dashboards (p. 18) .	October 8, 2015
Added AWS Lambda metrics and dimensions	Added AWS Lambda metrics and dimensions.	September 4, 2015
Added Amazon Elastic Container Service metrics and dimensions	Added Amazon Elastic Container Service metrics and dimensions.	August 17, 2015
Added Amazon Simple Storage Service metrics and dimensions	Added Amazon Simple Storage Service metrics and dimensions.	July 26, 2015
New feature: Reboot alarm action	Added the reboot alarm action and new IAM role for use with alarm actions. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance (p. 154) .	July 23, 2015
Added Amazon WorkSpaces metrics and dimensions	Added Amazon WorkSpaces metrics and dimensions.	April 30, 2015
Added Amazon Machine Learning metrics and dimensions	Added Amazon Machine Learning metrics and dimensions.	April 9, 2015
New feature: Amazon EC2 instance recovery alarm actions	Updated alarm actions to include new EC2 instance recovery action. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance (p. 154) .	March 12, 2015
Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions	Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions.	March 6, 2015
Added Amazon Simple Workflow Service metrics and dimensions	Added Amazon Simple Workflow Service metrics and dimensions.	May 9, 2014
Updated guide to add support for AWS CloudTrail	Added a new topic to explain how you can use AWS CloudTrail to log activity in Amazon CloudWatch. For more information, see Logging Amazon CloudWatch API calls with AWS CloudTrail (p. 958) .	April 30, 2014

Change	Description	Release date
Updated guide to use the new AWS Command Line Interface (AWS CLI)	<p>The AWS CLI is a cross-service CLI with a simplified installation, unified configuration, and consistent command line syntax. The AWS CLI is supported on Linux/Unix, Windows, and Mac. The CLI examples in this guide have been updated to use the new AWS CLI.</p> <p>For information about how to install and configure the new AWS CLI, see Getting Set Up with the AWS CLI Interface in the <i>AWS Command Line Interface User Guide</i>.</p>	February 21, 2014
Added Amazon Redshift and AWS OpsWorks metrics and dimensions	Added Amazon Redshift and AWS OpsWorks metrics and dimensions.	July 16, 2013
Added Amazon Route 53 metrics and dimensions	Added Amazon Route 53 metrics and dimensions.	June 26, 2013
New feature: Amazon CloudWatch Alarm Actions	Added a new section to document Amazon CloudWatch alarm actions, which you can use to stop or terminate an Amazon Elastic Compute Cloud instance. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance (p. 154) .	January 8, 2013
Updated EBS metrics	Updated the EBS metrics to include two new metrics for Provisioned IOPS volumes.	November 20, 2012
New billing alerts	You can now monitor your AWS charges using Amazon CloudWatch metrics and create alarms to notify you when you have exceeded the specified threshold. For more information, see Creating a billing alarm to monitor your estimated AWS charges (p. 159) .	May 10, 2012
New metrics	You can now access six new Elastic Load Balancing metrics that provide counts of various HTTP response codes.	October 19, 2011
New feature	You can now access metrics from Amazon EMR.	June 30, 2011
New feature	You can now access metrics from Amazon Simple Notification Service and Amazon Simple Queue Service.	July 14, 2011
New Feature	Added information about using the PutMetricData API to publish custom metrics. For more information, see Publishing custom metrics (p. 94) .	May 10, 2011
Updated metrics retention	Amazon CloudWatch now retains the history of an alarm for two weeks rather than six weeks. With this change, the retention period for alarms matches the retention period for metrics data.	April 7, 2011

Change	Description	Release date
New feature	Added ability to send Amazon Simple Notification Service or Auto Scaling notifications when a metric has crossed a threshold. For more information, see Alarms (p. 8) .	December 2, 2010
New feature	A number of CloudWatch actions now include the MaxRecords and NextToken parameters, which enable you to control pages of results to display.	December 2, 2010
New feature	This service now integrates with AWS Identity and Access Management (IAM).	December 2, 2010