



Designing and implementing logging and monitoring with Amazon CloudWatch

AWS Prescriptive Guidance



AWS Prescriptive Guidance: Designing and implementing logging and monitoring with Amazon CloudWatch

Copyright © 2024 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

Introduction	1
Targeted business outcomes	5
Accelerate operational readiness	5
Improve operational excellence	5
Enhance operational visibility	5
Scale operations and reduce overhead costs	6
Planning your CloudWatch deployment	7
Using CloudWatch in centralized or distributed accounts	8
Managing CloudWatch agent configuration files	11
Managing CloudWatch configurations	12
Example: Storing CloudWatch configuration files in an S3 bucket	14
Configuring the CloudWatch agent for EC2 instances and on-premises servers	16
Configuring the CloudWatch agent	16
Configuring log capture for EC2 instances	17
Configuring metrics capture for EC2 instances	19
System-level CloudWatch configuration	22
Configuring system-level logs	22
Configuring system-level metrics	24
Application-level CloudWatch configuration	25
Configuring application-level logs	25
Configuring application-level metrics	26
CloudWatch agent installation approaches for Amazon EC2 and on-premises servers	28
Installing the CloudWatch agent using Systems Manager Distributor and State Manager	28
Set up State Manager and Distributor for CloudWatch agent deployment and configuration	30
Use Systems Manager Quick Setup and manually update the created Systems Manager resources	32
Use AWS CloudFormation instead of Quick Setup	33
Customized Quick Setup in a single account and Region with an AWS CloudFormation stack	34
Customized Quick Setup in multiple Regions and multiple accounts with AWS CloudFormation StackSets	35
Considerations for configuring on-premises servers	36
Considerations for ephemeral EC2 instances	38

Using an automated solution to deploy the CloudWatch agent	39
Deploying the CloudWatch agent during instance provisioning with the user data script	39
Including the CloudWatch agent in your AMIs	40
Logging and monitoring on Amazon ECS	41
Configuring CloudWatch with an EC2 launch type	41
Amazon ECS container logs for EC2 and Fargate launch types	43
Using custom log routing with FireLens for Amazon ECS	44
Metrics for Amazon ECS	44
Creating custom application metrics in Amazon ECS	45
Logging and monitoring on Amazon EKS	47
Logging for Amazon EKS	47
Amazon EKS control plane logging	48
Amazon EKS node and application logging	48
Logging for Amazon EKS on Fargate	50
Metrics for Amazon EKS and Kubernetes	51
Kubernetes control plane metrics	51
Node and system metrics for Kubernetes	51
Application metrics	53
Metrics for Amazon EKS on Fargate	53
Prometheus monitoring on Amazon EKS	55
Logging and metrics for AWS Lambda	57
Lambda function logging	57
Sending logs to other destinations from CloudWatch	58
Lambda function metrics	59
System-level metrics	59
Application metrics	60
Searching and analyzing logs in CloudWatch	61
Collectively monitor and analyze applications with CloudWatch Application Insights	61
Performing log analysis with CloudWatch Logs Insights	64
Performing log analysis with Amazon OpenSearch Service	66
Alarming options with CloudWatch	69
Using CloudWatch alarms to monitor and alarm	69
Using CloudWatch anomaly detection to monitor and alarm	70
Alarming across multiple Regions and accounts	70
Automating alarm creation with EC2 instance tags	71
Monitoring application and service availability	72

Tracing applications with AWS X-Ray	73
Deploying the X-Ray daemon to trace applications and services on Amazon EC2	73
Deploying the X-Ray daemon to trace applications and services on Amazon ECS or Amazon EKS	74
Configuring Lambda to trace requests to X-Ray	74
Instrumenting your applications for X-Ray	75
Configuring the X-Ray sampling rules	75
Dashboards and visualizations with CloudWatch	77
Creating cross-service dashboards	77
Creating application or workload-specific dashboards	77
Creating cross-account or cross-Region dashboards	78
Using metric math to fine-tune observability and alarming	79
Using automatic dashboards for Amazon ECS, Amazon EKS, and Lambda with CloudWatchContainer Insights and CloudWatch Lambda Insights	79
CloudWatch integration with AWS services	81
Amazon Managed Grafana for dashboarding and visualization	82
FAQ	85
Where do I store my CloudWatch configuration files?	85
How can I create a ticket in my service management solution when an alarm is raised?	85
How do I use CloudWatch to capture log files in my containers?	85
How do I monitor health issues for AWS services?	86
How can I create a custom CloudWatch metric when no agent support exists?	86
How do I integrate my existing logging and monitoring tools with AWS?	86
Resources	87
Introduction	87
Targeted business outcomes	87
Planning your CloudWatch deployment	87
Configuring the CloudWatch agent for EC2 instances and on-premises servers	87
CloudWatch agent installation approaches for Amazon EC2 and on-premises servers	88
Logging and monitoring on Amazon ECS	88
Logging and monitoring on Amazon EKS	89
Logging and metrics for AWS Lambda	89
Searching and analyzing logs in CloudWatch	90
Alarming options with CloudWatch	90
Monitoring application and service availability	91
Tracing applications with AWS X-Ray	91

Dashboards and visualizations with CloudWatch	91
CloudWatch integration with AWS services	91
Amazon Managed Grafana for dashboarding and visualization	92
Document history	93
Glossary	94
#	94
A	95
B	98
C	99
D	102
E	106
F	108
G	109
H	110
I	111
L	113
M	114
O	118
P	120
Q	122
R	122
S	125
T	128
U	130
V	130
W	131
Z	132

Designing and implementing logging and monitoring with Amazon CloudWatch

Khurram Nizami, Amazon Web Services (AWS)

April 2023 ([document history](#))

This guide helps you design and implement logging and monitoring with [Amazon CloudWatch](#) and related Amazon Web Services (AWS) management and governance services for workloads that use [Amazon Elastic Compute Cloud \(Amazon EC2\) instances](#), [Amazon Elastic Container Service \(Amazon ECS\)](#), [Amazon Elastic Kubernetes Service \(Amazon EKS\)](#), [AWS Lambda](#), and on-premises servers. The guide is intended for operations teams, DevOps engineers, and application engineers that manage workloads on the AWS Cloud.

Your logging and monitoring approach should be based on the [six pillars](#) of the AWS Well-Architected Framework. These pillars are [operational excellence](#), [security](#), [reliability](#), [performance efficiency](#), and [cost optimization](#). A well-architected monitoring and alarming solution improves reliability and performance by helping you proactively analyze and adjust your infrastructure.

This guide doesn't extensively discuss logging and monitoring for security or cost-optimization because these are topics that require in-depth evaluation. There are many AWS services that support security logging and monitoring, including [AWS CloudTrail](#), [AWS Config](#), [Amazon Inspector](#), [Amazon Detective](#), [Amazon Macie](#), [Amazon GuardDuty](#), and [AWS Security Hub](#). You can also use [AWS Cost Explorer](#), [AWS Budgets](#), and [CloudWatch billing metrics](#) for cost optimization.

The following table outlines the six areas that your logging and monitoring solution should address.

Capturing and ingesting log files and metrics

Identify, configure, and send system and application logs and metrics to AWS services from different sources.

Searching and analyzing logs

Search and analyze logs for operations management, problem identification, troubleshooting, and applications analysis.

Monitoring metrics and alarming	Identify and act on observations and trends in your workloads.
Monitoring application and service availability	Reduce downtime and improve your ability to meet service level targets by continuously monitoring service availability.
Tracing applications	Trace application requests in systems and external dependencies to fine-tune performance, perform root cause analysis, and troubleshoot issues.
Creating dashboards and visualizations	Create dashboards that focus on relevant metrics and observations for your systems and workloads, which helps continuous improvement and proactive discovery of issues.

CloudWatch can meet most logging and monitoring requirements, and provides a reliable, scalable, and flexible solution. Many AWS services automatically provide CloudWatch metrics, in addition to CloudWatch logging integration for monitoring and analysis. CloudWatch also provides agents and log drivers to support a variety of compute options such as servers (both in the cloud and on premises), containers, and serverless computing. This guide also covers the following AWS services that are used with logging and monitoring:

- [AWS Systems Manager Distributor](#), [Systems Manager State Manager](#), and [Systems Manager Automation](#) to automate, configure, and update the CloudWatch agent for your EC2 instances and on-premises servers
- [Amazon OpenSearch Service](#) for advanced log aggregation, search, and analysis
- [Amazon Route 53 health checks](#) and [CloudWatch Synthetics](#) to monitor application and service availability
- [Amazon Managed Service for Prometheus](#) for monitoring containerized applications at scale
- [AWS X-Ray](#) for application tracing and runtime analysis
- [Amazon Managed Grafana](#) to visualize and analyze data from multiple sources (for example, CloudWatch, Amazon OpenSearch Service, and [Amazon Timestream](#))

The AWS compute services that you choose also affect the implementation and configuration of your logging and monitoring solution. For example, CloudWatch's implementation and configuration is different for Amazon EC2, Amazon ECS, Amazon EKS, and Lambda.

Application and workload owners can often forget about logging and monitoring or inconsistently configure and implement it. This means that workloads enter production with limited observability, which causes delays in identifying issues and increases the time taken to troubleshoot and resolve them. At a minimum, your logging and monitoring solution must address the systems layer for the operating system (OS)-level logs and metrics, in addition to the application layer for application logs and metrics. The guide provides a recommended approach for addressing these two layers across different compute types, including the three compute types outlined in the following table.

Long-running and immutable EC2 instances	System and application logs and metrics across multiple operating systems (OSs) in multiple AWS Regions or accounts.
Containers	System and application logs and metrics for your Amazon ECS and Amazon EKS clusters, including examples for different configurations.
Serverless	System and application logs and metrics for your Lambda functions and considerations for customization.

This guide provides a logging and monitoring solution that addresses CloudWatch and related AWS services in the following areas:

- [Planning your CloudWatch deployment](#) – Considerations for planning your CloudWatch deployment and guidance on centralizing your CloudWatch configuration.
- [Configuring the CloudWatch agent for EC2 instances and on-premises servers](#) – CloudWatch configuration details for system-level and application-level logging and metrics.
- [CloudWatch agent installation approaches for Amazon EC2 and on-premises servers](#) – Approaches for installing the CloudWatch agent, including automated deployment using Systems Manager across multiple Regions and accounts.

- [Logging and monitoring on Amazon ECS](#) – Guidance for configuring CloudWatch for cluster-level and application-level logging and metrics in Amazon ECS.
- [Logging and monitoring on Amazon EKS](#) – Guidance for configuring CloudWatch for cluster-level and application-level logging and metrics in Amazon EKS.
- [Prometheus monitoring on Amazon EKS](#) – Introduces and compares Amazon Managed Service for Prometheus with CloudWatch Container Insights monitoring for Prometheus.
- [Logging and metrics for AWS Lambda](#) – Guidance for configuring CloudWatch for your Lambda functions.
- [Searching and analyzing logs in CloudWatch](#) – Methods to analyze your logs using Amazon CloudWatch Application Insights, CloudWatch Logs Insights, and extending log analysis to Amazon OpenSearch Service.
- [Alarming options with CloudWatch](#) – Introduces CloudWatch Alarms and CloudWatch Anomaly Detection and provides guidance on alarm creation and setup.
- [Monitoring application and service availability](#) – Introduces and compares CloudWatch Synthetics and Route 53 health checks for automated availability monitoring.
- [Tracing applications with AWS X-Ray](#) – Introduction and setup for application tracing using X-Ray for Amazon EC2, Amazon ECS, Amazon EKS, and Lambda
- [Dashboards and visualizations with CloudWatch](#) – Introduction to CloudWatch Dashboards for improved observability across AWS workloads.
- [CloudWatch integration with AWS services](#) – Explains how CloudWatch integrates with various AWS services.
- [Amazon Managed Grafana for dashboarding and visualization](#) – Introduces and compares Amazon Managed Grafana with CloudWatch for dashboarding and visualization.

Implementation examples are used throughout this guide across these areas and are also available from the [AWS Samples GitHub repository](#).

Targeted business outcomes

Creating a logging and monitoring solution designed for the AWS Cloud is integral for achieving the [six advantages of cloud computing](#). Your logging and monitoring solution should help your IT organization achieve business outcomes that benefit your business processes, business partners, employees, and customers. You can expect the following four outcomes after implementing a logging and monitoring solution aligned with the [AWS Well-Architected Framework](#):

Accelerate operational readiness

Enabling a logging and monitoring solution is an important component of preparing a workload for production support and use. Operational readiness can quickly become a bottleneck if you rely too heavily on manual processes and can also reduce the time to value (TTV) for your IT investments. An ineffective approach also results in limited observability of your workloads. This can increase the risk of prolonged outages, customer dissatisfaction, and failed business processes.

You can use this guide's approaches to standardize and automate your logging and monitoring on the AWS Cloud. New workloads then require minimal manual preparation and intervention for production logging and monitoring. This also helps reduce the time and steps required to create logging and monitoring standards at scale for different workloads across multiple accounts and Regions.

Improve operational excellence

This guide provides multiple best practices for logging and monitoring that help diverse workloads meet business objectives and [operational excellence](#). This guide also provides [detailed examples and open-source, reusable templates](#) that you can use with an infrastructure as code (IaC) approach to implement a well-architected logging and monitoring solution using AWS services. Improving operational excellence is iterative and requires continuous improvement. The guide provides suggestions on how to continuously improve logging and monitoring practices.

Enhance operational visibility

Your business processes and applications might be supported by different IT resources and hosted on different compute types, either on premises or on the AWS Cloud. Your operational visibility can be limited by inconsistent and incomplete implementations of your logging and monitoring

strategy. Adopting a comprehensive logging and monitoring approach helps you quickly identify, diagnose, and respond to issues across your workloads. This guide helps you design and implement approaches to improve your complete operational visibility and reduce the mean time to resolve (MTTR) failures. A comprehensive logging and monitoring approach also helps your organization improve service quality, enhance end-user experience, and meet service-level agreements (SLAs).

Scale operations and reduce overhead costs

You can scale the logging and monitoring practices from this guide to support multiple Regions and accounts, short-lived resources, and multiple environments. The guide provides approaches and examples to automate manual steps (for example installing and configuring agents, monitoring metrics, and notifying or taking action when issues occur). These approaches are helpful when your cloud adoption matures and grows and you need to scale operational capability without increasing cloud management activities or resources.

Planning your CloudWatch deployment

The complexity and scope of a logging and monitoring solution depends on several factors, including:

- How many environments, Regions, and accounts are used and how this number might increase.
- The variety and types of your existing workloads and architectures.
- The compute types and OSs that must be logged and monitored.
- Whether there are both on-premises locations and AWS infrastructure.
- The aggregation and analytic requirements of multiple systems and applications.
- Security requirements that prevent unauthorized exposure of logs and metrics.
- Products and solutions that must integrate with your logging and monitoring solution to support operational processes.

You must regularly review and update your logging and monitoring solution with new or updated workload deployments. Updates to your logging, monitoring, and alarming should be identified and applied when issues are observed. These issues can then be proactively identified and prevented in the future.

You must make sure that you consistently install and configure software and services for capturing and ingesting logs and metrics. An established logging and monitoring approach uses multiple AWS or independent software vendor (ISV) services and solutions for different domains (for example, security, performance, networking, or analytics). Each domain has its own deployment and configuration requirements.

We recommend using CloudWatch to capture and ingest logs and metrics for multiple OSs and compute types. Many AWS services use CloudWatch to log, monitor, and publish logs and metrics, without requiring further configuration. CloudWatch provides a [software agent](#) that can be installed and configured for different OSs and environments. The following sections outline how to deploy, install, and configure the CloudWatch agent for multiple accounts, Regions, and configurations:

Topics

- [Using CloudWatch in centralized or distributed accounts](#)
- [Managing CloudWatch agent configuration files](#)

Using CloudWatch in centralized or distributed accounts

Although CloudWatch is designed to monitor AWS services or resources in one account and Region, you can use a central account to capture logs and metrics from multiple accounts and Regions. If you use more than one account or Region, you should evaluate whether to use the centralized account approach or an individual account to capture logs and metrics. Typically, a hybrid approach is required for multi-account and multi-Region deployments to support the requirements of security, analytics, operations, and workload owners.

The following table provides areas to consider when choosing to use a centralized, distributed, or hybrid approach.

Account structures	Your organization might have several separate accounts (for example, accounts for non-production and production workloads) or thousands of accounts for single applications in specific environments. We recommend that you maintain application logs and metrics in the account that the workload runs on, which gives workload owners access to the logs and metrics. This enables them to have an active role in logging and monitoring. We also recommend that you use a separate logging account to aggregate all workload logs for analysis, aggregation, trends, and centralized operations. Separate logging accounts can also be used for security, archiving and monitoring, and analytics.
Access requirements	Team members (for example, workload owners or developers) require access to logs and metrics to troubleshoot and make improvements. Logs should be maintained in the workload's account to make access and troubleshooting easier. If logs and metrics are maintained in a separate account from the workload, users might need to regularly alternate between accounts. Using a centralized account provides log information to authorized users without granting access to the workload account. This can simplify access requirements for analytic workloads where aggregation is required from workloads running in multiple accounts. The centralized logging account can also have alternati

	<p>ve search and aggregation options, such as an Amazon OpenSearch Service cluster. Amazon OpenSearch Service provides fine-grained access control down to the field level for your logs. Fine-grained access control is important when you have sensitive or confidential data that requires specialized access and permissions.</p>
Operations	<p>Many organizations have a centralized operations and security team or an external organization for operational support that requires access to logs for monitoring. Centralized logging and monitoring can make it easier to identify trends, search, aggregate, and perform analytics across all accounts and workloads. If your organization uses the “you build it, you run it” approach for DevOps, then workload owners require logging and monitoring information in their account. A hybrid approach might be required to satisfy central operations and analytics, in addition to distributed workload ownership.</p>
Environment	<p>You can choose to host logs and metrics in a central location for production accounts and keep logs and metrics for other environments (for example, development or testing) in the same or separate accounts, depending on security requirements and account architecture. This helps prevent sensitive data created during production from being accessed by a broader audience.</p>

CloudWatch provides [multiple options](#) to process logs in real time with CloudWatch subscription filters. You can use subscription filters to stream logs in real time to AWS services for custom processing, analysis, and loading to other systems. This can be particularly helpful if you take a hybrid approach where your logs and metrics are available in individual accounts and Regions, in addition to a centralized account and Region. The following list provides examples of AWS services that can be used for this:

- [Amazon Data Firehose](#) – Firehose provides a streaming solution that automatically scales and resizes based on the data volume being produced. You don’t need to manage the number of shards in an Amazon Kinesis data stream and you can directly connect to Amazon Simple

Storage Service (Amazon S3), Amazon OpenSearch Service, or Amazon Redshift with no additional coding. Firehose is an effective solution if you want to centralize your logs in those AWS services.

- [Amazon Kinesis Data Streams](#) – Kinesis Data Streams is an appropriate solution if you need to integrate with a service that Firehose doesn't support and implement additional processing logic. You can create an Amazon CloudWatch Logs destination in your accounts and Regions that specifies a Kinesis data stream in a central account and an AWS Identity and Access Management (IAM) role that grants it permission to place records in the stream. Kinesis Data Streams provides a flexible, open-ended landing zone for your log data that can then be consumed by different options. You can read the Kinesis Data Streams log data into your account, perform preprocessing, and send the data to your chosen destination.

However, you must configure the shards for the stream so that it is appropriately sized for the log data that is produced. Kinesis Data Streams acts as a temporary intermediary or queue for your log data, and you can store the data within the Kinesis stream for between one to 365 days. Kinesis Data Streams also supports replay capability, which means you can replay data that was not consumed.

- [Amazon OpenSearch Service](#) – CloudWatch Logs can stream logs in a log group to an OpenSearch cluster in an individual or centralized account. When you configure a log group to stream data to an OpenSearch cluster, a Lambda function is created in the same account and Region as your log group. The Lambda function must have a network connection with the OpenSearch cluster. You can customize the Lambda function to perform additional preprocessing, in addition to customizing the ingestion into Amazon OpenSearch Service. Centralized logging with Amazon OpenSearch Service makes it easier to analyze, search, and troubleshoot issues across multiple components in your cloud architecture.
- [Lambda](#) – If you use Kinesis Data Streams, you need to provision and manage compute resources that consume data from your stream. To avoid this, you can stream log data directly to Lambda for processing and send it to a destination based on your logic. This means that you don't need to provision and manage compute resources to process incoming data. If you choose to use Lambda, make sure that your solution is compatible with [Lambda quotas](#).

You might need to process or share log data stored in CloudWatch Logs in file format. You can create an export task to [export a log group to Amazon S3](#) for a specific date or time range. For example, you might choose to export logs on a daily basis to Amazon S3 for analytics and auditing. Lambda can be used to automate this solution. You can also combine this solution with Amazon S3

replication to ship and centralize your logs from multiple accounts and Regions to one centralized account and Region.

The CloudWatch agent configuration can also specify a `credentials` field in the [agent section](#). This specifies an IAM role to use when sending metrics and logs to a different account. If specified, this field contains the `role_arn` parameter. This field can be used when you only need centralized logging and monitoring in a specific centralized account and Region.

You can also use [AWS SDK](#) to write your own custom processing application in a language of your choice, read logs and metrics from your accounts, and send data to a centralized account or other destination for further processing and monitoring.

Managing CloudWatch agent configuration files

We recommend that you create a standard Amazon CloudWatch agent configuration that includes the system logs and metrics that you want to capture across all your Amazon Elastic Compute Cloud (Amazon EC2) instances and on-premises servers. You can use the CloudWatch agent [configuration file wizard](#) to help you create the configuration file. You can run the configuration wizard multiple times to generate unique configurations for different systems and environments. You can also modify the configuration file or create variations by [using the configuration file schema](#). The CloudWatch agent configuration file can be stored in [AWS Systems Manager Parameter Store](#) parameters. You can create separate Parameter Store parameters if you have [multiple CloudWatch agent configuration files](#). If you are using multiple AWS accounts or AWS Regions, you must manage and update the Parameter Store parameters in each account and Region. Alternatively, you can centrally manage your CloudWatch configurations as files in Amazon S3 or a version-control tool of your choice.

The `amazon-cloudwatch-agent-ctl` script included with the CloudWatch agent allows you to specify a configuration file, Parameter Store parameter, or the agent's default configuration. The default configuration aligns to the basic, predefined metric set and configures the agent to report memory and disk space metrics to CloudWatch. However, it doesn't include any log file configurations. The default configuration is also applied if you use [Systems Manager Quick Setup](#) for the CloudWatch agent.

Because the default configuration doesn't include logging and isn't customized for your requirements, we recommend that you create and apply your own CloudWatch configurations, customized to your requirements.

Managing CloudWatch configurations

By default, CloudWatch configurations can be stored and applied as Parameter Store parameters or as CloudWatch configuration files. The best choice will depend on your requirements. In this section, we discuss the pros and cons for these two options. A representative solution is also detailed for managing CloudWatch configuration files for multiple AWS accounts and AWS Regions.

Systems Manager Parameter Store parameters

Using Parameter Store parameters to manage CloudWatch configurations works well if you have a single, standard CloudWatch agent configuration file that you want to apply and manage in a small set of AWS accounts and Regions. When you store your CloudWatch configurations as Parameter Store parameters, you can use the CloudWatch agent configuration tool (`amazon-cloudwatch-agent-ctl` on Linux) to read and apply the configuration from Parameter Store without requiring you to copy the configuration file to your instance. You can use the **AmazonCloudWatch-ManageAgent** Systems Manager Command document to update the CloudWatch configuration on multiple EC2 instances in a single run. Because Parameter Store parameters are regional, you must update and maintain your CloudWatch Parameter Store parameters in each AWS Region and AWS account. If you have multiple CloudWatch configurations that you want to apply to each instance, you must customize the **AmazonCloudWatch-ManageAgent** Command document to include these parameters.

CloudWatch configuration files

Managing your CloudWatch configurations as files might work well if you have many AWS accounts and Regions and you are managing multiple CloudWatch configuration files. Using this approach, you can browse, organize, and manage them in a folder structure. You can apply security rules to individual folders or files to limit and grant access such as update and read permissions. You can share and transfer them outside of AWS for collaboration. You can version control the files to track and manage changes. You can apply CloudWatch configurations collectively by copying the configuration files to the CloudWatch agent configuration directory without applying each configuration file individually. For Linux, the CloudWatch configuration directory is found at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d`. For Windows, the configuration directory is found at `C:\ProgramData\Amazon\AmazonCloudWatchAgent\Configs`.

When you start the CloudWatch agent, the agent automatically appends each file found in these directories to create a CloudWatch composite configuration file. The configuration files should

be stored in a central location (for example, an S3 bucket) that can be accessed by your required accounts and Regions. An example solution using this approach is provided.

Organizing CloudWatch configurations

Regardless of the approach used to manage your CloudWatch configurations, organize your CloudWatch configurations. You can organize your configurations into file or Parameter Store paths using an approach such as the following.

/config/standard/windows/ec2

Store standard Windows-specific CloudWatch configuration files for Amazon EC2. You can further categorize your standard operating system (OS) configurations for different Windows versions, EC2 instance types, and environments under this folder.

/config/standard/windows/onpremises

Store standard Windows-specific CloudWatch configuration files for on-premises servers. You also further categorize your standard OS configurations for different Windows versions, server types, and environments under this folder.

/config/standard/linux/ec2

Store your standard Linux-specific CloudWatch configuration files for Amazon EC2. You can further categorize your standard OS configuration for different Linux distributions, EC2 instance types, and environments under this folder.

/config/standard/linux/onpremises

Store your standard Linux-specific CloudWatch configuration files for on-premises servers. You can further categorize your standard OS configuration for different Linux distributions, server types, and environments under this folder.

/config/ecs

Store CloudWatch configuration files that are specific to Amazon Elastic Container Service (Amazon ECS) if you use Amazon ECS container instances. These configurations can be appended to the standard Amazon EC2 configurations for Amazon ECS specific systems-level logging and monitoring.

/config/<application_name>

Store your application-specific CloudWatch configuration files. You can further categorize your applications with additional folders and prefixes for environments and versions.

Example: Storing CloudWatch configuration files in an S3 bucket

This section provides an example using Amazon S3 to store CloudWatch configuration files and a custom Systems Manager runbook to retrieve and apply the CloudWatch configuration files. This approach can address some of the challenges of using Systems Manager Parameter Store parameters for CloudWatch configuration at scale:

- If you use multiple Regions, you must synchronize CloudWatch configuration updates in each Region's Parameter Store. Parameter Store is a Regional service and the same parameter must be updated in each Region that uses the CloudWatch agent.
- If you have multiple CloudWatch configurations, you must initiate the retrieval and application of each Parameter Store configuration. You must individually retrieve each CloudWatch configuration from the Parameter Store and also update the retrieval method whenever you add a new configuration. In contrast, CloudWatch provides a configuration directory for storing configuration files and applies each configuration in the directory, without requiring them to be individually specified.
- If you use multiple accounts, you must ensure that each new account has the required CloudWatch configurations in its Parameter Store. You also need to make sure that any configuration changes are applied to these accounts and their Regions in the future.

You can store CloudWatch configurations in an S3 bucket that is accessible from all your accounts and Regions. You can then copy these configurations from the S3 bucket to the CloudWatch

configuration directory by using Systems Manager Automation runbooks and Systems Manager State Manager. You can use the [cloudwatch-config-s3-bucket.yaml](#) AWS CloudFormation template to create an S3 bucket that is accessible from multiple accounts within an organization in AWS Organizations. The template includes an `OrganizationID` parameter that grants read access to all accounts within your [organization](#).

The augmented sample Systems Manager runbook, provided in the [Set up State Manager and Distributor for CloudWatch agent deployment and configuration](#) section of this guide, is configured to retrieve files using the S3 bucket created by the [cloudwatch-config-s3-bucket.yaml](#) AWS CloudFormation template.

Alternatively, you can use a version control system (for example, GitHub or [AWS CodeCommit](#)) to store your configuration files. If you want to automatically retrieve configuration files stored in a version control system, you have to manage or centralize the credential storage and update the Systems Manager Automation runbook that is used to retrieve the credentials across your accounts and Regions.

Configuring the CloudWatch agent for EC2 instances and on-premises servers

Many organizations run workloads on both physical servers and virtual machines (VMs). These workloads typically run on different OSs that each have unique installation and configuration requirements for capturing and ingesting metrics.

If you choose to use EC2 instances, you can have a high level of control over your instance and OS configuration. However, this higher level of control and responsibility requires you to monitor and adjust configurations to achieve more efficient usage. You can improve your operational effectiveness by establishing standards for logging and monitoring, and applying a standard installation and configuration approach for capturing and ingesting logs and metrics.

Organizations that migrate or extend their IT investments to the AWS Cloud can leverage CloudWatch to achieve a unified logging and monitoring solution. CloudWatch pricing means that you incrementally pay for the metrics and logs that you want to capture. You can also capture logs and metrics for on-premises servers by using a similar CloudWatch agent installation process as that for Amazon EC2.

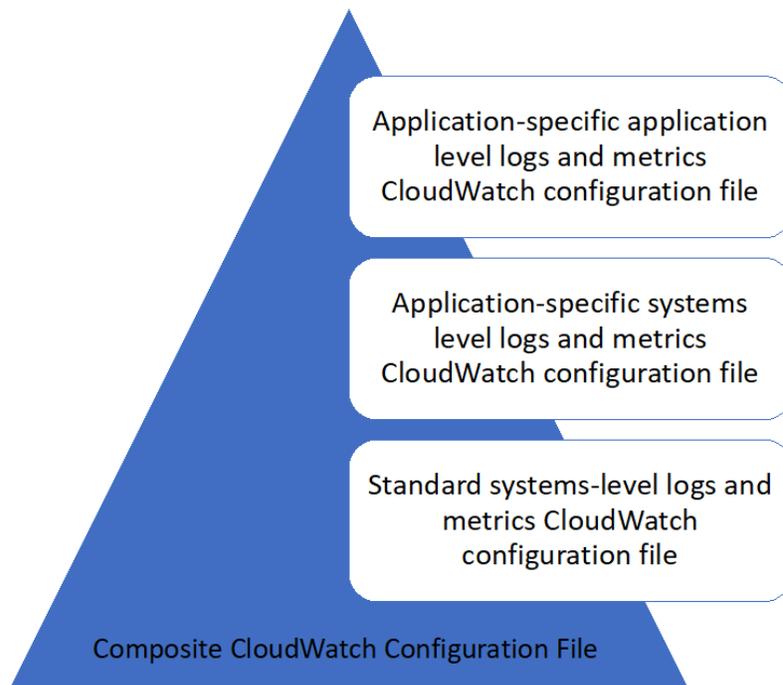
Before you begin installing and deploying CloudWatch, make sure that you evaluate the logging and metric configurations for your systems and applications. Ensure that you define the standard logs and metrics that you need to capture for the OSs that you want to use. System logs and metrics are the foundation and standard for a logging and monitoring solution because they are generated by the OS and are different for Linux and Windows. There are important metrics and log files available across Linux distributions, in addition to those that are specific to a Linux version or distribution. This variance also occurs between different Windows versions.

Configuring the CloudWatch agent

CloudWatch captures metrics and logs for Amazon EC2 and on-premises servers by using [CloudWatch agents and agent configuration files](#) that are specific to each OS. We recommend that you define your organization's standard metric and log capture configuration before you begin installing the CloudWatch agent at scale in your accounts.

You can combine multiple CloudWatch agent configurations to form a composite CloudWatch agent configuration. One recommended approach is to define and divide configurations for your logs and metrics at the system and application level. The following diagram illustrates how

multiple CloudWatch configuration file types for different requirements can be combined to form a composite CloudWatch configuration:



These logs and metrics can also be further classified and configured for specific environments or requirements. For example, you could define a smaller subset of logs and metrics with lower precision for unregulated development environments, and a larger, more complete set with higher precision for regulated production environments.

Configuring log capture for EC2 instances

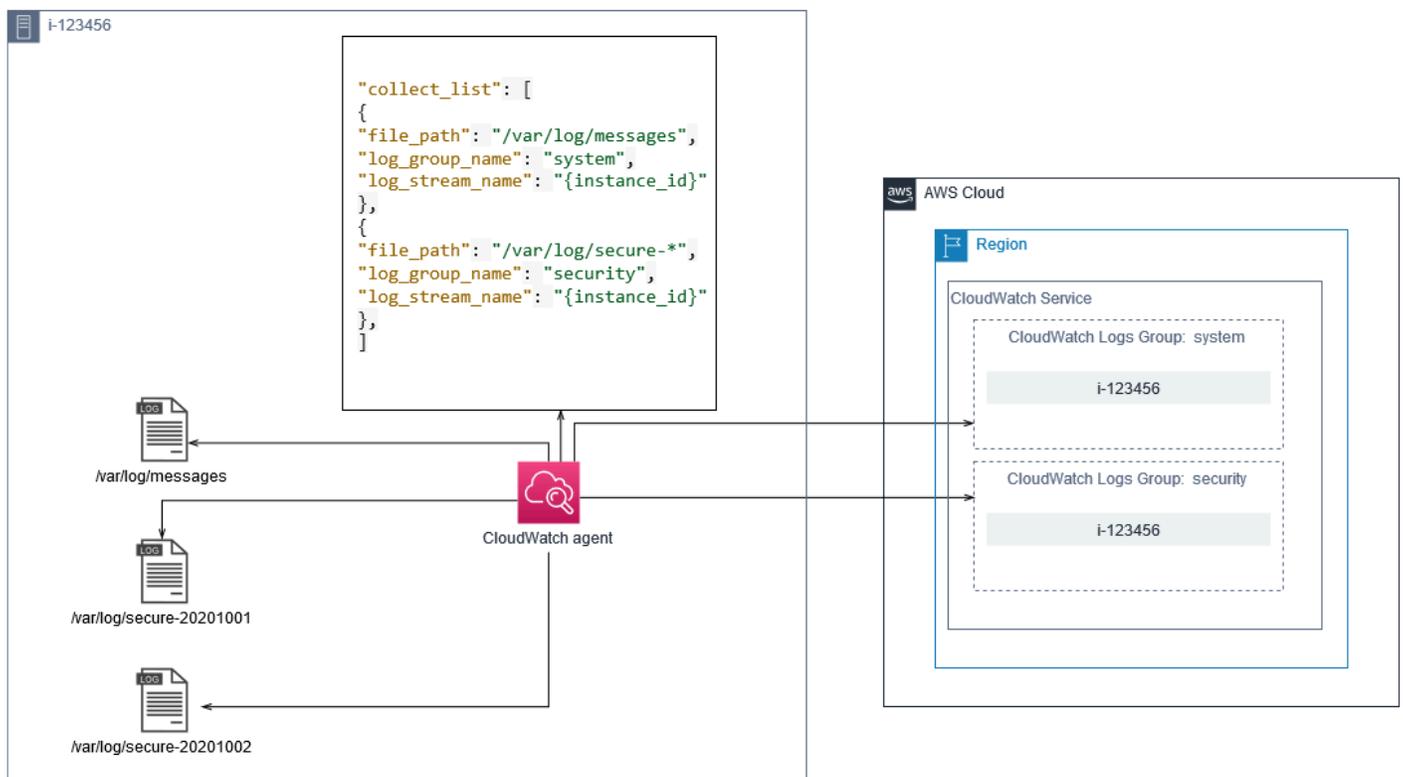
By default, Amazon EC2 doesn't monitor or capture log files. Instead, log files are captured and ingested into CloudWatch Logs by the CloudWatch agent software installed on your EC2 instance, AWS API, or AWS Command Line Interface (AWS CLI). We recommend using the CloudWatch agent to ingest log files into CloudWatch Logs for Amazon EC2 and on-premises servers.

You can search and filter logs, as well as extract metrics and run automation based on pattern patching from log files in CloudWatch. CloudWatch supports plaintext, space delimited, and JSON-formatted filter and pattern syntax options, with JSON-formatted logs providing the most flexibility. To increase the filtering and analysis options, you should use a formatted log output instead of plain text.

The CloudWatch agent uses a configuration file that defines the logs and metrics to send to CloudWatch. CloudWatch then captures each log file as a [log stream](#) and groups these log streams into a [log group](#). This helps you perform operations across logs from your EC2 instances, such as searching for a matching string.

The default log stream name is the same as the EC2 instance ID and the default log group name is the same as the log file path. The log stream's name must be unique within the CloudWatch log group. You can use `instance_id`, `hostname`, `local_hostname`, or `ip_address` for dynamic substitution in the log stream and log group names, which means that you can use the same CloudWatch agent configuration file across multiple EC2 instances.

The following diagram shows a CloudWatch agent configuration for capturing logs. The log group is defined by the captured log files and contains separate log streams for each EC2 instance because the `{instance_id}` variable is used for the log stream name and EC2 instance IDs are unique.



Log groups define the retention, tags, security, metric filters, and search scope for the log streams that they contain. The default grouping behavior based on the log file name helps you search, create metrics, and alarm on data that is specific to a log file across EC2 instances in an account and Region. You should evaluate whether further log group refinement is required. For example,

your account might be shared by multiple business units and have different technical or operations owners. This means that you must further refine the log group name to reflect the separation and ownership. This approach allows you to concentrate your analysis and troubleshooting on the relevant EC2 instance.

If multiple environments use one account, you can separate the logging for workloads that run in each environment. The following table shows a log group naming convention that includes the business unit, project or application, and environment.

Log group name	<code>/<Business unit>/<Project or application name>/<Environment>/<Log file name></code>
Log stream name	<code><EC2 instance ID></code>

You can also group all log files for an EC2 instance into the same log group. This makes it easier to search and analyze across a set of log files for a single EC2 instance. This is useful if most of your EC2 instances service one application or workload and each EC2 instance serves a specific purpose. The following table shows how your log group and log stream naming could be formatted to support this approach.

Log group name	<code>/<Business unit>/<Project or application name>/<Environment>/<EC2 instance ID></code>
Log stream name	<code><Log file name></code>

Configuring metrics capture for EC2 instances

By default, your EC2 instances are enabled for basic monitoring and a [standard set of metrics](#) (for example, CPU, network, or storage-related metrics) is automatically sent to CloudWatch every five minutes. CloudWatch metrics can vary depending on the instance family, for example, [burstable performance instances](#) have metrics for CPU credits. Amazon EC2 standard metrics are included in your instance price. If you enable [detailed monitoring](#) for your EC2 instances, you can

receive data in one-minute periods. The period frequency impacts your CloudWatch costs, so make sure that you evaluate whether detailed monitoring is required for all or only some of your EC2 instances. For example, you could enable detailed monitoring for production workloads but use basic monitoring for non-production workloads.

On-premises servers don't include any default metrics for CloudWatch and must use the CloudWatch agent, AWS CLI, or AWS SDK to capture metrics. This means that you must define the metrics that you want to capture (for example, CPU utilization) in the CloudWatch configuration file. You can create a unique CloudWatch configuration file that includes the standard EC2 instance metrics for your on-premises servers and apply it in addition to your standard CloudWatch configuration.

[Metrics](#) in CloudWatch are uniquely defined by metric name and zero or more dimensions, and are uniquely grouped in a metric namespace. Metrics provided by an AWS service have a namespace that begins with AWS (for example, AWS/EC2), and non-AWS metrics are considered custom metrics. Metrics that you configure and capture with the CloudWatch agent are all considered custom metrics. Because the number of created metrics impacts your CloudWatch costs, you should evaluate whether each metric is required for all or only some of your EC2 instances. For example, you could define a complete set of metrics for production workloads but use a smaller subset of these metrics for non-production workloads.

CWAgent is the default namespace for metrics published by the CloudWatch agent. Similar to log groups, the metric namespace organizes a set of metrics so that they can be found together in one place. You should modify the namespace to reflect a business unit, project or application, and environment (for example, /<Business unit>/<Project or application name>/<Environment>). This approach is useful if multiple unrelated workloads use the same account. You can also correlate your namespace naming convention to your CloudWatch log group naming convention.

Metrics are also identified by their dimensions, which help you analyze them against a set of conditions and are the properties that observations are recorded against. Amazon EC2 includes [separate metrics](#) for EC2 instances with InstanceId and AutoScalingGroupName dimensions. You also receive metrics with the ImageId and InstanceType dimensions if you enable detailed monitoring. For example, Amazon EC2 provides a separate EC2 instance metric for the CPU utilization with the InstanceId dimensions, in addition to separate CPU utilization metric for the InstanceType dimension. This helps you analyze CPU utilization for each unique EC2 instance, in addition to all EC2 instances of a specific [instance type](#).

Adding more dimensions increases your analysis capability but also increases your overall costs, because each metric and unique dimension value combination results in a new metric. For example, if you create a metric for the memory utilization percentage against the InstanceId dimension, then this is a new metric for each EC2 instance. If your organization runs thousands of EC2 instances, this causes thousands of metrics and results in higher costs. To control and predict costs, make sure that you determine the metric's cardinality and which dimensions add the most value. For example, you could define a complete set of dimensions for your production workload metrics but a smaller subset of these dimensions for non-production workloads.

You can use the `append_dimensions` property to add dimensions to one or all metrics defined in your CloudWatch configuration. You can also dynamically append the `ImageId`, `InstanceId`, `InstanceType`, and `AutoScalingGroupName` to all metrics in your CloudWatch configuration. Alternatively, you can append an arbitrary dimension name and value for specific metrics by using the `append_dimensions` property on that metric. CloudWatch can also aggregate statistics on metric dimensions that you defined with the `aggregation_dimensions` property.

For example, you could aggregate the memory used against the `InstanceType` dimension to see the average memory used by all EC2 instances for each instance type. If you use `t2.micro` instances running in a Region, you could determine if workloads using the `t2.micro` class are overutilizing or underutilizing the memory provided. Underutilization might be a sign of workloads using EC2 classes with an unrequired memory capacity. In contrast, overutilization might be a sign of workloads using Amazon EC2 classes with insufficient memory.

The following diagram shows a sample CloudWatch metrics configuration that uses a custom namespace, added dimensions, and aggregation by `InstanceType`.



System-level CloudWatch configuration

Systems-level metrics and logs are a central component of a monitoring and logging solution, and the CloudWatch agent has specific configuration options for Windows and Linux.

We recommend that you use the [CloudWatch configuration file wizard](#) or configuration file schema to define the CloudWatch agent configuration file for each OS that you plan to support. Additional workload-specific, OS-level logs and metrics can be defined in separate CloudWatch configuration files and appended to the standard configuration. These unique configuration files should be separately stored in an S3 bucket where they can be retrieved by your EC2 instances. An example of an S3 bucket setup for this purpose is described in the [Managing CloudWatch configurations](#) section of this guide. You can automatically retrieve and apply these configurations using State Manager and Distributor.

Configuring system-level logs

System-level logs are essential for diagnosing and troubleshooting issues on premises or on the AWS Cloud. Your log capture approach should include any system and security logs generated by the OS. The OS-generated log files might be different depending on the OS version.

The CloudWatch agent supports monitoring Windows event logs by providing the event log name. You can choose which Windows event logs you want to monitor (for example System, Application, or Security).

The system, application, and security logs for Linux systems are typically stored in the `/var/log` directory. The following table defines the common default log files that you should monitor, but you should check the `/etc/rsyslog.conf` or `/etc/syslog.conf` file to determine the specific setup for your system's log files.

Fedora distribution

(Amazon Linux, CentOS, Red Hat Enterprise Linux)

`/var/log/boot.log*` – Bootup log

`/var/log/dmesg` – Kernel log

`/var/log/secure` – Security and authentication log

`/var/log/messages` – General system log

`/var/log/cron*` – Cron Logs

	<code>/var/log/cloud-init-output.log</code> – Output from Userdata startup scripts
Debian	<code>/var/log/syslog</code> – Bootup log
(Ubuntu)	<code>/var/log/cloud-init-output.log</code> – Output from Userdata startup scripts
	<code>/var/log/auth.log</code> – Security and authentication log
	<code>/var/log/kern.log</code> – Kernel log

Your organization might also have other agents or system components that generate logs you want to monitor. You should evaluate and decide which log files are generated by these agents or applications, and include them in your configuration by identifying their file location. For example, you should include the Systems Manager and CloudWatch agent logs in your configuration. The following table provides the location of these agent logs for Windows and Linux.

Windows	CloudWatch agent	<code>\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log</code>
	Systems Manager agent	<code>%PROGRAMDATA%\Amazon\SSM\Logs\amazon-ssm-agent.log</code> <code>%PROGRAMDATA%\Amazon\SSM\Logs\errors.log</code> <code>%PROGRAMDATA%\Amazon\SSM\Logs\audits\amazon-ssm-agent-audit-YYYY-MM-DD</code>

Linux	CloudWatch agent	<code>/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log</code>
	Systems Manager agent	<code>/var/log/amazon/ssm/amazon-ssm-agent.log</code> <code>/var/log/amazon/ssm/errors.log</code> <code>/var/log/amazon/ssm/audits/amazon-ssm-agent-audit-YYYY-MM-DD</code>

CloudWatch ignores a log file if the log file is defined in the CloudWatch agent configuration but isn't found. This is useful when you want to maintain a single log configuration for Linux, instead of separate configurations for each distribution. It is also useful when a log file doesn't exist until the agent or software application starts running.

Configuring system-level metrics

Memory and disk space utilization aren't included in standard metrics provided by Amazon EC2. To include these metrics, you must install and configure the CloudWatch agent on your EC2 instances. The CloudWatch agent configuration wizard creates a CloudWatch configuration with [predefined metrics](#) and you can add or remove metrics as required. Make sure that you review the predefined metric sets to determine the appropriate level that you require.

End users and workload owners should publish additional system metrics based on specific requirements for a server or EC2 instance. These metric definitions should be stored, versioned, and maintained in a separate CloudWatch agent configuration file, and shared in a central location (for example, Amazon S3) for reuse and automation.

Standard Amazon EC2 metrics are not automatically captured in on-premises servers. These metrics must be defined in a CloudWatch agent configuration file used by the on-premises instances. You can create a separate metric configuration file for on-premises instances with

metrics such as CPU utilization, and have these metrics appended to the standard metrics configuration file.

Application-level CloudWatch configuration

Application logs and metrics are generated by running applications and are application specific. Make sure that you define the logs and metrics required to adequately monitor applications that are regularly used by your organization. For example, your organization might have standardized on Microsoft Internet Information Server (IIS) for web-based applications. You can create a standard log and metric CloudWatch configuration for IIS that can also be used across your organization. Application-specific configuration files can be stored in a centralized location (for example, an S3 bucket) and are accessed by workload owners or through automated retrieval, and copied to the CloudWatch configuration directory. The CloudWatch agent automatically combines CloudWatch configuration files found in the configuration file directory of each EC2 instance or server into a composite CloudWatch configuration. The end result is a CloudWatch configuration that includes your organization's standard system-level configuration, as well as all relevant application-level CloudWatch configurations.

Workload owners should identify and configure log files and metrics for all critical applications and components.

Configuring application-level logs

Application-level logging varies depending on whether the application is a commercial off-the-shelf (COTS) or custom developed application. COTS applications and their components might provide several options for log configuration and output, such as log detail level, log file format, and log file location. However, most COTS or third-party applications don't allow you to fundamentally change the logging (for example, updating the application's code to include additional log statements or formats that are not configurable). At a minimum, you should configure logging options for COTS or third-party applications to log warning and error-level information, preferably in JSON format.

You can integrate custom-developed applications with CloudWatch Logs by including the application's log files in your CloudWatch configuration. Custom applications provide better log quality and control because you can customize the log output format, categorize and separate component output to separate log files, in addition to including any additional required details. Make sure that you review and standardize on logging libraries and the required data and formatting for your organization so that analytics and processing become easier.

You can also write to a CloudWatch log stream with the CloudWatch Logs [PutLogEvents](#) API call or by using the AWS SDK. You can use the API or SDK for custom logging requirements, such as coordinating logging to a single log stream across a distributed set of components and servers. However, the easiest to maintain and most widely applicable solution is to configure your applications to write to log files and then use the CloudWatch agent to read and stream the log files to CloudWatch.

You should also consider the kind of metrics that you want to measure from your application log files. You can use metric filters to measure, graph, and alarm on this data in a CloudWatch log group. For example, you can use a metric filter to count failed login attempts by identifying them in your logs.

You can also create custom metrics for your custom-developed applications by using the [CloudWatch embedded metric format](#) in your application log files.

Configuring application-level metrics

Custom metrics are metrics that aren't directly provided by AWS services to CloudWatch and they are published in a custom namespace in CloudWatch metrics. All application metrics are considered custom CloudWatch metrics. Application metrics might align to an EC2 instance, application component, API call, or even a business function. You must also consider the importance and cardinality of the dimensions that you choose for your metrics. Dimensions with high cardinality generate a large number of custom metrics and could increase your CloudWatch costs.

CloudWatch helps you capture application-level metrics in multiple ways, including the following:

- Capture process-level metrics by defining the individual processes that you want to capture from the [procstat plugin](#).
- An application publishes a metric to Windows Performance Monitor and this metric is defined in the CloudWatch configuration.
- Metric filters and patterns are applied against an application's logs in CloudWatch.
- An application writes to a CloudWatch log by using the CloudWatch embedded metric format.
- An application sends a metric to CloudWatch through the API or AWS SDK.
- An application sends a metric to a [collectd](#) or [StatsD](#) daemon with a configured CloudWatch agent.

You can use `procstat` to monitor and measure critical application processes with the CloudWatch agent. This helps you to raise an alarm and take action (for example, a notification or restart process) if a critical process is no longer running for your application. You can also measure the performance characteristics of your application processes and raise an alarm if a particular process is acting abnormally.

Procstat monitoring is also useful if you can't update your COTS applications with additional custom metrics. For example, you can create a `my_process` metric that measures the `cpu_time` and includes a custom `application_version` dimension. You can also use multiple CloudWatch agent configuration files for an application if you have different dimensions for different metrics.

If your application runs on Windows, you should evaluate if it already publishes metrics to Windows Performance Monitor. Many COTS applications integrate with Windows Performance Monitor, which helps you easily monitor application metrics. CloudWatch also integrates with Windows Performance Monitor and you can capture any metrics that are already available in it.

Make sure that you review the logging format and log information provided by your applications to determine which metrics can be extracted with metric filters. You could review historical logs for the application to determine how error messages and abnormal shutdowns are represented. You should also review previously reported issues to determine if a metric could be captured to prevent the issue from recurring. You should also review the application's documentation and ask the application developers to confirm how error messages can be identified.

For custom-developed applications, work with the application's developers to define important metrics that can be implemented by using the CloudWatch embedded metric format, AWS SDK, or AWS API. The recommended approach is to use the embedded metric format. You can use the AWS provided open-source embedded metric format libraries to help you write your statements in the required format. You would also need to update your [application-specific CloudWatch configuration](#) to include the embedded metric format agent. This causes the agent running on the EC2 instance to act as a local embedded metric format endpoint that sends embedded metric format metrics to CloudWatch.

If your applications already support publishing metrics to `collectd` or `statsd`, you can leverage them to ingest metrics into CloudWatch.

CloudWatch agent installation approaches for Amazon EC2 and on-premises servers

Automating the CloudWatch agent's installation process helps you quickly and consistently deploy it and capture the required logs and metrics. There are several approaches for automating the CloudWatch agent installation, including multi-account and multi-Region support. The following automated installation approaches are discussed:

- [Installing the CloudWatch agent using Systems Manager Distributor and Systems Manager State Manager](#) – We recommend using this approach if your EC2 instances and on-premises servers are running the Systems Manager agent. This ensures that the CloudWatch agent is kept updated and you can report on and remediate servers that don't have the CloudWatch agent. This approach also scales to support multiple accounts and Regions.
- [Deploying the CloudWatch agent as a part of the user data script during EC2 instance provisioning](#) – Amazon EC2 allows you to define a startup script that is run when you first boot or reboot. You can define a script to automate the agent's download and installation process. This can also be included in AWS CloudFormation scripts and AWS Service Catalog products. This approach might be appropriate on an as-needed basis if there is a customized agent installation and configuration approach for a specific workload that deviates from your standards.
- [Including the CloudWatch agent in Amazon Machine Images \(AMIs\)](#) – You can install the CloudWatch agent in your custom AMIs for Amazon EC2. The EC2 instances that use the AMI will automatically have the agent installed and started. However, you must ensure the agent and its configuration are regularly updated.

Installing the CloudWatch agent using Systems Manager Distributor and State Manager

You can use Systems Manager State Manager with Systems Manager Distributor to automatically install and update the CloudWatch agent on servers and EC2 instances. Distributor includes the AmazonCloudWatchAgent AWS managed package that installs the most recent CloudWatch agent version.

This installation approach has the following prerequisites:

- The Systems Manager agent must be installed and running on your servers or EC2 instances. The Systems Manager agent is preinstalled on Amazon Linux, Amazon Linux 2, and some AMIs. The agent must also be installed and configured on other images or on-premises VMs and servers.
- An IAM role or credentials that have the [required CloudWatch and Systems Manager permissions](#) must be attached to the EC2 instance or defined in the credentials file for an on-premises server. For example, you can create an IAM role that includes the AWS managed policies: AmazonSSMManagedInstanceCore for Systems Manager and CloudWatchAgentServerPolicy for CloudWatch. You can use the [ssm-cloudwatch-instance-role.yaml](#) AWS CloudFormation template to deploy an IAM role and instance profile that includes both of these policies. This template can also be modified to include other standard IAM permissions for your EC2 instances. For on-premises servers or VMs, should configure the CloudWatch agent to use the [Systems Manager service role](#) that was configured for the on-premises server. For more information about this, see [How can I configure on-premises servers that use Systems Manager Agent and the unified CloudWatch agent to use only temporary credentials?](#) in the AWS Knowledge Center.

The following list provides several advantages for using the Systems Manager Distributor and State Manager approach to install and maintain the CloudWatch agent:

- **Automated installation for multiple OSs** – You don't need to write and maintain a script for each OS to download and install the CloudWatch agent.
- **Automatic update checks** – State Manager automatically and regularly checks that each EC2 instance has the most recent CloudWatch version.
- **Compliance reporting** – The Systems Manager compliance dashboard shows which EC2 instances failed to successfully install the Distributor package.
- **Automated installation for newly launched EC2 instances** – New EC2 instances that are launched into your account automatically receive the CloudWatch agent.

However, you should also consider the following three areas before you choose this approach:

- **Collision with an existing association** – If another association already installs or configures the CloudWatch agent, then the two associations might interfere with each other and potentially cause issues. When using this approach, you should remove any existing associations that install or update the CloudWatch agent and configuration.

- **Updating custom agent configuration files** – Distributor performs an installation by using the default configuration file. If you use a custom configuration file or multiple CloudWatch configuration files, you must update the configuration after the installation.
- **Multi-Region or multi-account setup** – The State Manager association must be set up in each account and Region. New accounts in a multi-account environment must be updated to include the State Manager association. You need to centralize or synchronize the CloudWatch configuration so that multiple accounts and Regions can retrieve and apply your required standards.

Set up State Manager and Distributor for CloudWatch agent deployment and configuration

You can use [Systems Manager Quick Setup](#) to quickly configure Systems Manager features, including automatically installing and updating the CloudWatch agent on your EC2 instances. The Quick Setup deploys an AWS CloudFormation stack that deploys and configures Systems Manager resources based on your choices.

The following list provides two important actions that are performed by Quick Setup for automated CloudWatch agent installation and update:

1. **Create Systems Manager custom documents** – Quick Setup creates the following Systems Manager documents for use with State Manager. The document names might vary but the content remains the same:
 - `CreateAndAttachIAMToInstance` – Creates the `AmazonSSMRoleForInstancesQuickSetup` role and instance profile if they don't exist and attaches the `AmazonSSMManagedInstanceCore` policy to the role. This doesn't include the required `CloudWatchAgentServerPolicy` IAM policy. You must update this policy and update this Systems Manager document to include this policy as described in the following section.
 - `InstallAndManageCloudWatchDocument` – Installs the CloudWatch agent with Distributor and configures each EC2 instance one time with a default CloudWatch agent configuration using the `AWS-ConfigureAWSPackage` Systems Manager document.
 - `UpdateCloudWatchDocument` – Updates the CloudWatch agent by installing the latest CloudWatch agent using the `AWS-ConfigureAWSPackage` Systems Manager document.

Updating or uninstalling the agent doesn't remove the existing CloudWatch configuration files from the EC2 instance.

2. **Create State Manager associations** – State Manager associations are created and configured to use the custom created Systems Manager documents. The State Manager association names might vary but the configuration remains the same:

- `ManageCloudWatchAgent` – Runs the `InstallAndManageCloudWatchDocument` Systems Manager document one time for each EC2 instance.
- `UpdateCloudWatchAgent` – Runs the `UpdateCloudWatchDocument` Systems Manager document every 30 days for each EC2 instance.
- Runs the `CreateAndAttachIAMToInstance` Systems Manager document one time for each EC2 instance.

You must augment and customize the completed Quick Setup configuration to include CloudWatch permissions and support custom CloudWatch configurations. In particular, the `CreateAndAttachIAMToInstance` and the `InstallAndManageCloudWatchDocument` document will need to be updated. You can manually update the Systems Manager documents created by Quick Setup. Alternatively, you can use your own CloudFormation template to provision the same resources with the necessary updates as well as configure and deploy other Systems Manager resources and not use Quick Setup.

Important

Quick Setup creates an AWS CloudFormation stack to deploy and configure Systems Manager resources based on your choices. If you update your Quick Setup choices, you might need to manually re-update the Systems Manager documents.

The following sections describe how to manually update the Systems Manager resources created by Quick Setup, as well as use your own AWS CloudFormation template to perform an updated Quick Setup. We recommend that you use your own AWS CloudFormation template to avoid manually updating resources created by Quick Setup and AWS CloudFormation.

Use Systems Manager Quick Setup and manually update the created Systems Manager resources

The Systems Manager resources created by the Quick Setup approach must be updated to include the required CloudWatch agent permissions and support multiple CloudWatch configuration files. This section describes how to update the IAM role and Systems Manager documents to use a centralized S3 bucket containing CloudWatch configurations that is accessible from multiple accounts. Creating an S3 bucket to store the CloudWatch configuration files is discussed in the [Managing CloudWatch configurations](#) section of this guide.

Update the `CreateAndAttachIAMToInstance` Systems Manager document

This Systems Manager document created by Quick Setup checks whether an EC2 instance has an existing IAM instance profile attached to it. If it does, it attaches the `AmazonSSMManagedInstanceCore` policy to the existing role. This protects your existing EC2 instances from losing AWS permissions that might be assigned through existing instance profiles. You need to add a step in this document to attach the `CloudWatchAgentServerPolicy` IAM policy to EC2 instances that already have an instance profile attached. The Systems Manager document also creates the IAM role if it doesn't exist and an EC2 instance doesn't have an instance profile attached to it. You must update this section of the document to also include the `CloudWatchAgentServerPolicy` IAM policy.

Review the completed [CreateAndAttachIAMToInstance.yaml](#) sample document and compare it to the document created by Quick Setup. Edit the existing document to include the required steps and changes. Based on your Quick Setup choices the document created by Quick Setup might be different than the provided sample document, so ensure that you make the required adjustments. The sample document includes the Quick Setup option choice to scan instances for missing patches daily and therefore includes a policy for Systems Manager Patch Manager.

Update the `InstallAndManageCloudWatchDocument` Systems Manager document

This Systems Manager document created by Quick Setup installs the CloudWatch agent and configures it with the default CloudWatch agent configuration. The default CloudWatch configuration aligns to the basic, predefined metric set. You must replace the default configuration step and add steps to download your CloudWatch configuration files from your CloudWatch configuration S3 bucket.

Review the completed [InstallAndManageCloudWatchDocument.yaml](#) updated document and compare it to the document created by Quick Setup. The document created by your Quick Setup might be different, so make sure that you have made the required adjustments. Edit your existing document to include the necessary steps and changes.

Use AWS CloudFormation instead of Quick Setup

Instead of using Quick Setup, you can use AWS CloudFormation to configure Systems Manager. This approach allows you to customize your Systems Manager configuration according to your specific requirements. This approach also avoids manual updates to the configured Systems Manager resources created by Quick Setup to support custom CloudWatch configurations.

The Quick Setup feature also uses AWS CloudFormation and creates a AWS CloudFormation stack set to deploy and configure Systems Manager resources based on your choices. Before you can use AWS CloudFormation stack sets, you must create the IAM roles used by AWS CloudFormation StackSets to support deployments across multiple accounts or Regions. Quick Setup creates the roles it requires to support multi-Region or multi-account deployments with AWS CloudFormation StackSets. You must complete the prerequisites for AWS CloudFormation StackSets if you want to configure and deploy Systems Manager resources in multiple Regions or multiple accounts from a single account and Region. For more information about this, see [Prerequisites for stack set operations](#) in the AWS CloudFormation documentation.

Review the [AWS-QuickSetup-SSMHostMgmt.yaml](#) AWS CloudFormation template for customized Quick Setup.

You should review the resources and capabilities in the AWS CloudFormation template and make adjustments according to your requirements. You should version control the AWS CloudFormation template that you use and incrementally test changes to confirm the required result. Additionally, you should perform cloud security reviews to determine if there are any policy adjustments that are required based on your organization's requirements.

You should deploy the AWS CloudFormation stack in a single test account and Region, and perform any necessary test cases to customize and confirm the desired result. You can then graduate your deployment to multiple Regions in a single account, and then to multiple accounts and multiple regions.

Customized Quick Setup in a single account and Region with an AWS CloudFormation stack

If you are only use a single account and Region, you can deploy the complete example as a AWS CloudFormation stack instead of an AWS CloudFormation stack set. However if possible, we recommend that you use the multi-account, multi-Region stack set approach even if only use a single account and Region. Using AWS CloudFormation StackSets makes it easier to expand to additional accounts and Regions in the future.

Use the following steps to deploy the [AWS-QuickSetup-SSMHostMgmt.yaml](#) AWS CloudFormation template as an AWS CloudFormation stack in a single account and Region:

1. Download the template and check it into your preferred version control system (for example, AWS CodeCommit).
2. Customize the default AWS CloudFormation parameter values based on your organization's requirements.
3. Customize the State Manager association schedules.
4. Customize the Systems Manager document with the `InstallAndManageCloudWatchDocument` logical ID. Confirm that the S3 bucket prefixes align to the prefixes for the S3 bucket containing your CloudWatch configuration.
5. Retrieve and record the Amazon Resource Name (ARN) for the S3 bucket containing your CloudWatch configurations. For more information about this, see the [Managing CloudWatch configurations](#) section of this guide. A sample [cloudwatch-config-s3-bucket.yaml](#) AWS CloudFormation template is available that includes a bucket policy to provide read access to AWS Organizations accounts.
6. Deploy the customized Quick Setup AWS CloudFormation template to the same account as your S3 bucket:
 - For the `CloudWatchConfigBucketARN` parameter, enter the S3 bucket's ARN.
 - Make adjustments to the parameter options depending on the capabilities that you want to enable for Systems Manager.
7. Deploy a test EC2 instance with and without an IAM role to confirm that the EC2 instance works with CloudWatch.

- Apply the `AttachIAMToInstance` State Manager association. This is a Systems Manager runbook that is configured to run on a schedule. State Manager associations that use runbooks are not automatically applied to new EC2 instances and can be configured to run on a scheduled basis. For more information, see [Running automations with triggers using State Manager](#) in the Systems Manager documentation.
- Confirm that the EC2 instance has the required IAM role attached.
- Confirm that the Systems Manager agent is working correctly by confirming that the EC2 instance is visible in Systems Manager.
- Confirm that the CloudWatch agent is working correctly by viewing CloudWatch logs and metrics based on the CloudWatch configurations from your S3 bucket.

Customized Quick Setup in multiple Regions and multiple accounts with AWS CloudFormation StackSets

If you are using multiple accounts and Regions, then you can deploy the [AWS-QuickSetup-SSMHostMgmt.yaml](#) AWS CloudFormation template as a stack set. You must complete the [AWS CloudFormation StackSet prerequisites](#) before using stack sets. The requirements vary depending on whether you are deploying stack sets with [self-managed or service-managed permissions](#).

We recommend that you deploy stack sets with service-managed permissions so that new accounts automatically receive the customized Quick Setup. You must deploy a service-managed stack set from the AWS Organizations management account or delegated administrator account. You should deploy the stack set from a centralized account used for automation that has delegated administrator privileges, rather than the AWS Organizations management account. We also recommend that you test your stack set deployment by targeting a test organizational unit (OU) with a single or small number of accounts in one Region.

1. Complete steps 1 to 5 from the [Customized Quick Setup in a single account and Region with an AWS CloudFormation stack](#) section of this guide.
2. Sign in to the AWS Management Console, open the AWS CloudFormation console and choose **Create StackSet**:
 - Choose **Template is ready** and **Upload a template file**. Upload the AWS CloudFormation template that you customized to your requirements.
 - Specify the stack set details:
 - Enter a stack set name, for example, `StackSet-SSM-QuickSetup`.

- Make adjustments to the parameter options depending on the capabilities that you want to enable for Systems Manager.
- For the `CloudWatchConfigBucketARN` parameter, enter the ARN for your CloudWatch configuration's S3 bucket.
- Specify the stack set options, choose whether you will use service-managed permissions with AWS Organizations or self-managed permissions.
 - If you choose self-managed permissions, enter the **`AWSCloudFormationStackSetAdministrationRole`** and **`AWSCloudFormationStackSetExecutionRole`** IAM role details. The administrator role must exist in the account and the execution role must exist in each target account
- For **service-managed** permissions with AWS Organizations, we recommend that you first deploy to a test OU instead of the entire organization.
 - Choose whether you want to enable automatic deployments. We recommend that you choose **Enabled**. For account removal behavior, the recommended setting is **Delete stacks**.
- For **self-managed** permissions, enter the AWS account IDs for the accounts that you want to set up. You must repeat this process for each new account if you use self-managed permissions.
- Enter the Regions where you will be using CloudWatch and Systems Manager.
- Confirm that the deployment is successful by viewing the status in the **Operations and Stack instances** tab for the stack set.
- Test that Systems Manager and CloudWatch are correctly working in the deployed accounts by following step 7 from the [Customized Quick Setup in a single account and Region with an AWS CloudFormation stack](#) section of this guide.

Considerations for configuring on-premises servers

The CloudWatch agent for on-premises servers and VMs is installed and configured by using a similar approach to that for EC2 instances. However, the following table provides considerations that you must evaluate when installing and configuring the CloudWatch agent on on-premises servers and VMs.

Point the CloudWatch agent to the same temporary credentials used for Systems Manager.

When you set up Systems Manager in a hybrid environment that includes on-premises servers, you can activate Systems Manager with an IAM role. You should use the role created for your EC2 instances that includes the `CloudWatchAgentServerPolicy` and `AmazonSSMManagedInstanceCore` policies.

This results in the Systems Manager agent retrieving and writing temporary credentials to a local credentials file. You can point your CloudWatch agent configuration to the same file. You can use the process from [Configure on-premises servers that use Systems Manager agent and the unified CloudWatch agent to use only temporary credentials](#) in the AWS Knowledge Center.

You can also automate this process by defining a separate Systems Manager Automation runbook and State Manager association, and targeting your on-premises instances with tags. When you create an [Systems Manager activation](#) for your on-premises instances, you should include a tag that identifies the instances as on-premises instances.

Consider using accounts and Regions that have VPN or AWS Direct Connect access and AWS PrivateLink.

You can use AWS Direct Connect or AWS Virtual Private Network (AWS VPN) to establish private connections between on-premises networks and your virtual private cloud (VPC). AWS PrivateLink establishes a private connection to CloudWatch Logs with an interface VPC endpoint. This approach is

useful if you have restrictions that prevent data being sent over the public internet to a public service endpoint.

All metrics must be included in the CloudWatch configuration file.

Amazon EC2 includes standard metrics (for example, CPU utilization) but these metrics must be defined for on-premises instances. You can use a separate platform configuration file to define these metrics for on-premises servers and then append the configuration to the standard CloudWatch metrics configuration for the platform.

Considerations for ephemeral EC2 instances

EC2 instances are temporary, or *ephemeral*, if they are provisioned by Amazon EC2 Auto Scaling, Amazon EMR, [Amazon EC2 Spot Instances](#), or AWS Batch. Ephemeral EC2 instances can cause a very large number of CloudWatch streams under a common log group without additional information on their runtime origin.

If you use ephemeral EC2 instances, consider adding additional dynamic contextual information in the log group and log stream names. For example, you can include the Spot Instance request ID, Amazon EMR cluster name, or Auto Scaling group name. This information can vary for newly launched EC2 instances and you might have to retrieve and configure it at runtime. You can do this by writing a CloudWatch agent configuration file at boot and restarting the agent to include the updated configuration file. This enables delivery of logs and metrics to CloudWatch using dynamic runtime information.

You should also make sure that your metrics and logs are sent by the CloudWatch agent before your ephemeral EC2 instances are terminated. The CloudWatch agent includes a `flush_interval` parameter that can be configured to define the time interval for flushing log and metric buffers. You can lower this value based on your workload and stop the CloudWatch agent and force the buffers to flush before the EC2 instance is terminated.

Using an automated solution to deploy the CloudWatch agent

If you use an automation solution (for example, Ansible or Chef), you can leverage it to automatically install and update the CloudWatch agent. If you use this approach, you must evaluate the following considerations:

- **Validate that the automation covers the OSs and the OS versions that you support.** If the automation script doesn't support all your organization's OSs, you should define alternative solutions for the unsupported OSs.
- **Validate that the automation solution regularly checks for CloudWatch agent updates and upgrades.** Your automation solution should regularly check for updates to the CloudWatch agent, or regularly uninstall and reinstall the agent. You can use a scheduler or automation solution functionality to regularly check and update the agent.
- **Validate that you can confirm agent installation and configuration compliance.** Your automation solution should enable you to determine when a system doesn't have the agent installed or when the agent isn't working. You can implement a notification or alarm into your automation solution so that failed installations and configurations are tracked.

Deploying the CloudWatch agent during instance provisioning with the user data script

You can use this approach if you don't plan to use Systems Manager and want to selectively use CloudWatch for your EC2 instances. Typically, this approach is used on a one-time basis or when a specialized configuration is required. AWS provides [direct links](#) for the CloudWatch agent that can be downloaded in your start-up or user data scripts. The agent installation packages can be silently run without user interaction, which means that you can use them in automated deployments. If you use this approach, you should evaluate the following considerations:

- **Increased risk that users won't install the agent or configure standard metrics.** Users might provision instances without including the necessary steps to install the CloudWatch agent. They could also misconfigure the agent, which might cause logging and monitoring inconsistencies.
- **The installation scripts must be OS specific and suitable for different OS versions.** You require separate scripts if you intended to use both Windows and Linux. The Linux script should also have different installation steps based on the distribution.

- **You must regularly update the CloudWatch agent with new versions when available.** This can be automated if you use Systems Manager with State Manager, but you can also configure the user data script to rerun on instance startup. The CloudWatch agent is then updated and reinstalled on every reboot.
- **You must automate the retrieval and application of standard CloudWatch configurations.** This can be automated if you use Systems Manager with State Manager, but you can also configure a user data script to retrieve the configuration files on boot and restart the CloudWatch agent.

Including the CloudWatch agent in your AMIs

The advantage of using this approach is that you don't have to wait for the CloudWatch agent to be installed and configured, and you can immediately begin logging and monitoring. This helps you better monitor your instance provisioning and startup steps in case instances fail to start. This approach is also appropriate if you don't plan to use the Systems Manager agent. If you use this approach, you should evaluate the following considerations:

- **An update process must exist because AMIs might not include the most recent CloudWatch agent version.** The CloudWatch agent installed in an AMI is only current to the last time the AMI was created. You should include an additional method for updating the agent on a regular basis and when the EC2 instance is provisioned. If you use Systems Manager, you can use the [Installing the CloudWatch agent using Systems Manager Distributor and State Manager](#) solution provided in this guide for this. If you don't use Systems Manager, you can use a user data script to update the agent on instance startup and reboot.
- **Your CloudWatch agent configuration file must be retrieved on instance startup.** If you don't use Systems Manager, you can configure a user data script to retrieve the configuration files on boot and then restart the CloudWatch agent.
- **The CloudWatch agent must be restarted after your CloudWatch configuration is updated.**
- **AWS credentials must not be saved in the AMI.** Make sure that no local AWS credentials are stored in the AMI. If you use Amazon EC2, you can apply the necessary IAM role to your instance and avoid local credentials. If you use on-premises instances, you should automate or manually update the instance credentials before starting the CloudWatch agent.

Logging and monitoring on Amazon ECS

Amazon Elastic Container Service (Amazon ECS) provides [two launch types](#) for running containers and that determine the type of infrastructure that host tasks and services; these launch types are AWS Fargate and Amazon EC2. Both launch types integrate with CloudWatch but configurations and support vary.

The following sections help you understand how to use CloudWatch for logging and monitoring on Amazon ECS.

Topics

- [Configuring CloudWatch with an EC2 launch type](#)
- [Amazon ECS container logs for EC2 and Fargate launch types](#)
- [Using custom log routing with FireLens for Amazon ECS](#)
- [Metrics for Amazon ECS](#)

Configuring CloudWatch with an EC2 launch type

With an EC2 launch type, you provision an Amazon ECS cluster of EC2 instances that use the CloudWatch agent for logging and monitoring. An Amazon ECS optimized AMI comes pre-installed with the [Amazon ECS container agent](#) and provides CloudWatch metrics for the Amazon ECS cluster.

These default metrics are included in the cost of Amazon ECS, but the default configuration for Amazon ECS doesn't monitor log files or additional metrics (for example, free disk space). You can use the AWS Management Console to provision an Amazon ECS cluster with the EC2 launch type, this creates an AWS CloudFormation stack that deploys an Amazon EC2 Auto Scaling group with a launch configuration. However, this approach means that you can't choose a custom AMI or customize the launch configuration with different settings or additional boot up scripts.

To monitor additional logs and metrics, you must install the CloudWatch agent on your Amazon ECS container instances. You can use the installation approach for EC2 instances from the [Installing the CloudWatch agent using Systems Manager Distributor and State Manager](#) section of this guide. However, the Amazon ECS AMI doesn't include the required Systems Manager agent. You should use a custom launch configuration with a user data script that installs the Systems Manager agent

when you create your Amazon ECS cluster. This allows your container instances to register with Systems Manager and apply the State Manager associations to install, configure, and update the CloudWatch agent. When State Manager runs and updates your CloudWatch agent configuration, it also applies your standardized systems-level CloudWatch configuration for Amazon EC2. You can also store standardized CloudWatch configurations for Amazon ECS in the S3 bucket for your CloudWatch configuration and automatically apply them with State Manager.

You should make sure that the IAM role or instance profile applied to your Amazon ECS container instances includes the required `CloudWatchAgentServerPolicy` and `AmazonSSMManagedInstanceCore` policies. You can use the [ecs_cluster_with_cloudwatch_linux.yaml](#) AWS CloudFormation template to provision Linux-based Amazon ECS clusters. This template creates an Amazon ECS cluster with a custom launch configuration that installs Systems Manager and deploys a custom CloudWatch configuration to monitor log files specific to Amazon ECS.

You should capture the following logs for your Amazon ECS container instances, as well as your standard EC2 instance logs:

- **Amazon ECS agent startup output** – `/var/log/ecs/ecs-init.log`
- **Amazon ECS agent output** – `/var/log/ecs/ecs-agent.log`
- **IAM credential provider requests log** – `/var/log/ecs/audit.log`

For more information about output level, formatting, and additional configuration options, see [Amazon ECS log file locations](#) in the Amazon ECS documentation.

Important

Agent installation or configuration is not required for the Fargate launch type because you don't run or manage EC2 container instances.

Amazon ECS container instances should use the latest Amazon ECS optimized AMIs and container agent. AWS stores public Systems Manager Parameter Store parameters with Amazon ECS optimized AMI information, including the AMI ID. You can retrieve the latest most recently optimized AMI from the Parameter Store by using the [Parameter Store parameter format](#) for Amazon ECS optimized AMIs. You can refer to the public Parameter Store parameter that references the most recent AMI or a specific AMI release in your AWS CloudFormation templates.

AWS provides the same Parameter Store parameters in each supported Region. This means that AWS CloudFormation templates referencing these parameters can be reused across Regions and accounts without the AMI to be updated. You can control the deployment of newer Amazon ECS AMIs to your organization by referring to a specific release, which helps you prevent the use of a new Amazon ECS optimized AMI until you test it.

Amazon ECS container logs for EC2 and Fargate launch types

Amazon ECS uses a task definition to deploy and manage containers as tasks and services. You configure the containers that you want to launch into your Amazon ECS cluster within a task definition. Logging is configured with a log driver at the container level. Multiple log driver options provide your containers with different logging systems (for example, `awslogs`, `fluentd`, `gelf`, `json-file`, `journald`, `logentries`, `splunk`, `syslog`, or `awsfirelens`) depending on whether you use the EC2 or Fargate launch type. The Fargate launch type provides a subset of the following log driver options: `awslogs`, `splunk`, and `awsfirelens`. AWS provides the `awslogs` log driver to capture and transmit container output to CloudWatch Logs. Log driver settings enable you to customize the log group, Region, and log stream prefix along with many other options.

The default naming for log groups and the option used by the **Auto-configure CloudWatch Logs** option on the AWS Management Console is `/ecs/<task_name>`. The log stream name used by Amazon ECS has the `<awslogs-stream-prefix>/<container_name>/<task_id>` format. We recommend that you use a group name that groups your logs based on your organization's requirements. In the following table, the `image_name` and `image_tag` are included in the log stream's name.

Log group name	<code>/<Business unit>/<Project or application name>/<Environment>/<Cluster name>/<Task name></code>
Log stream name prefix	<code>/<image_name>/<image_tag></code>

This information is also available in the task definition. However, tasks are regularly updated with new revisions, which means that the task definition might have used a different `image_name` and `image_tag` than those that the task definition is currently using. For more information and naming suggestions, see the [Planning your CloudWatch deployment](#) section of this guide.

If you use a continuous integration and continuous delivery (CI/CD) pipeline or automated process, you can create a new task definition revision for your application with each new Docker image build. For example, you can include the Docker image name, image tag, GitHub revision, or other important information in your task definition revision and logging configuration as a part of your CI/CD process.

Using custom log routing with FireLens for Amazon ECS

FireLens for Amazon ECS helps you route logs to [Fluentd](#) or [Fluent Bit](#) so that you can directly send container logs to AWS services and AWS Partner Network (APN) destinations as well as support log shipping to CloudWatch Logs.

AWS provides a [Docker image for Fluent Bit](#) with pre-installed plugins for Amazon Kinesis Data Streams, Amazon Data Firehose, and CloudWatch Logs. You can use the FireLens log driver instead of the `awslogs` log driver for more customization and control over logs sent to CloudWatch Logs.

For example, you can use the FireLens log driver to control the log format output. This means that an Amazon ECS container's CloudWatch logs are automatically formatted as JSON objects and include JSON-formatted properties for `ecs_cluster`, `ecs_task_arn`, `ecs_task_definition`, `container_id`, `container_name`, and `ec2_instance_id`. The fluent host is exposed to your container through the `FLUENT_HOST` and `FLUENT_PORT` environment variables when you specify the `awsfirelens` driver. This means that you can directly log to the log router from your code by using fluent logger libraries. For example, your application might include the `fluent-logger-python` library to log to Fluent Bit by using the values available from the environment variables.

If you choose to use FireLens for Amazon ECS, you can configure the same settings as the `awslogs` log driver [and use other settings as well](#). For example, you can use the [ecs-task-nginx-firelense.json](#) Amazon ECS task definition that launches an NGINX server configured to use FireLens for logging to CloudWatch. It also launches a FireLens Fluent Bit container as a sidecar for logging.

Metrics for Amazon ECS

[Amazon ECS provides standard CloudWatch metrics](#) (for example, CPU and memory utilization) for the EC2 and Fargate launch types at the cluster and service level with the Amazon ECS container agent. You can also capture metrics for your services, tasks, and containers by using CloudWatch Container Insights, or capture your own custom container metrics by using the embedded metric format.

Container Insights is a CloudWatch feature that provides metrics such as CPU utilization, memory utilization, network traffic, and storage at the cluster, container instance, service, and task levels. Container Insights also creates automatic dashboards that help you analyze services and tasks, and see the average memory or CPU utilization at the container level. Container Insights publishes custom metrics to the ECS/ContainerInsights [custom namespace](#) that you can use for graphing, alarming, and dashboarding.

You can turn on Container Insight metrics by enabling Container Insights for each individual Amazon ECS cluster. If you also want to see metrics at the container instance level, you can [launch the CloudWatch agent as a daemon container on your Amazon ECS cluster](#). You can use the [cwagent-ecs-instance-metric-cfn.yaml](#) AWS CloudFormation template to deploy the CloudWatch agent as an Amazon ECS service. Importantly, this example assumes that you created an appropriate custom CloudWatch agent configuration and stored it in Parameter Store with the key `ecs-cwagent-daemon-service`.

The [CloudWatch agent](#) deployed as a daemon container for CloudWatch Container Insights includes additional disk, memory, and CPU metrics such as `instance_cpu_reserved_capacity` and `instance_memory_reserved_capacity` with the `ClusterName`, `ContainerInstanceId`, `InstanceId` dimensions. Metrics at the container instance level are implemented by Container Insights by using the CloudWatch embedded metric format. You can configure additional system-level metrics for your Amazon ECS container instances by using the approach from from the [Set up State Manager and Distributor for CloudWatch agent deployment and configuration](#) section of this guide.

Creating custom application metrics in Amazon ECS

You can create custom metrics for your applications by using the [CloudWatch embedded metric format](#). The `awslogs` log driver can interpret CloudWatch embedded metric format statements.

The `CW_CONFIG_CONTENT` environment variable in the following example is set to the contents of the `cwagentconfig` Systems Manager Parameter Store parameter. You can run the agent with this basic configuration to configure it as an embedded metric format endpoint. However, it is no longer necessary.

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
```

```
}  
}  
}
```

If you have Amazon ECS deployments across multiple accounts and Regions, you can use an AWS Secrets Manager secret to store your CloudWatch configuration and configure the secret policy to share it with your organization. You can use the `secrets` option in your task definition to set the `CW_CONFIG_CONTENT` variable.

You can use the AWS provided [open-source embedded metric format libraries](#) in your application and specify the `AWS_EMF_AGENT_ENDPOINT` environment variable to connect to your CloudWatch agent sidecar container acting as an embedded metric format endpoint. For example, you can use the [ecs_cw_emf_example](#) sample Python application to send metrics in embedded metric format to a CloudWatch agent sidecar container configured as an embedded metric format endpoint.

The [Fluent Bit plugin](#) for CloudWatch can also be used to send embedded metric format messages. You can also use the [ecs_firelense_emf_example](#) sample Python application to send metrics in embedded metric format to a Firelens for Amazon ECS sidecar container.

If you don't want to use embedded metric format, you can create and update CloudWatch metrics through the [AWS API](#) or AWS [SDK](#). We don't recommend this approach unless you have a specific use case, because it adds maintenance and management overhead to your code.

Logging and monitoring on Amazon EKS

Amazon Elastic Kubernetes Service (Amazon EKS) integrates with CloudWatch Logs for the Kubernetes control plane. The control plane is provided as a managed service by Amazon EKS and you can [turn on logging without installing a CloudWatch agent](#). The CloudWatch agent can also be deployed to capture Amazon EKS node and container logs. [Fluent Bit and Fluentd](#) are also supported for sending your container logs to CloudWatch Logs.

CloudWatch Container Insights provides a comprehensive metrics monitoring solution for Amazon EKS at the cluster, node, pod, task, and service level. Amazon EKS also supports multiple options for metrics capture with [Prometheus](#). The Amazon EKS control plane [provides a metrics endpoint](#) that exposes metrics in a Prometheus format. You can deploy Prometheus into your Amazon EKS cluster to consume these metrics.

You can also [set up the CloudWatch agent to scrape Prometheus metrics](#) and create CloudWatch metrics, in addition to consume other Prometheus endpoints. [Container Insights monitoring for Prometheus](#) can also automatically discover and capture Prometheus metrics from supported, containerized workloads and systems.

You can install and configure the CloudWatch agent on your Amazon EKS nodes, in a similar way to the approach used for Amazon EC2 with Distributor and State Manager, to align your Amazon EKS nodes with your standard system logging and monitoring configurations.

Logging for Amazon EKS

Kubernetes logging can be divided into control plane logging, node logging, and application logging. The [Kubernetes control plane](#) is a set of components that manage Kubernetes clusters and produce logs used for auditing and diagnostic purposes. With Amazon EKS, you can [turn on logs for different control plane components](#) and send them to CloudWatch.

Kubernetes also runs system components such as `kubelet` and `kube-proxy` on each Kubernetes node that runs your pods. These components write logs within each node and you can configure CloudWatch and Container Insights to capture these logs for each Amazon EKS node.

Containers are grouped as [pods](#) within a Kubernetes cluster and are scheduled to run on your Kubernetes nodes. Most containerized applications write to standard output and standard error, and the container engine redirects the output to a logging driver. In Kubernetes, the container

logs are found in the `/var/log/pods` directory on a node. You can configure CloudWatch and Container Insights to capture these logs for each of your Amazon EKS pods.

Amazon EKS control plane logging

An Amazon EKS cluster consists of a high availability, single-tenant control plane for your Kubernetes cluster and the Amazon EKS nodes that run your containers. The control plane nodes run in an account managed by AWS. The Amazon EKS cluster control plane nodes are integrated with CloudWatch and you can turn on logging for specific control plane components.

Logs are provided for each Kubernetes control plane component instance. AWS manages the health of your control plane nodes and provides a [service-level agreement \(SLA\) for the Kubernetes endpoint](#).

Amazon EKS node and application logging

We recommend that you use [CloudWatch Container Insights](#) to capture logs and metrics for Amazon EKS. Container Insights implements cluster, node, and pod-level metrics with the CloudWatch agent, and Fluent Bit or Fluentd for log capture to CloudWatch. Container Insights also provides automatic dashboards with layered views of your captured CloudWatch metrics. Container Insights is deployed as CloudWatch DaemonSet and Fluent Bit DaemonSet that runs on every Amazon EKS node. Fargate nodes are not supported by Container Insights because the nodes are managed by AWS and don't support DaemonSets. Fargate logging for Amazon EKS is covered separately in this guide.

The following table shows the CloudWatch log groups and logs captured by the [default Fluentd or Fluent Bit log capture configuration](#) for Amazon EKS.

`/aws/containerinsights/Cluster_Name/
application`

All log files in `/var/log/containers`. This directory provides symbolic links to all the Kubernetes container logs in the `/var/log/pods` directory structure. This captures your application container logs writing to `stdout` or `stderr`. It also includes logs for Kubernetes system containers such as `aws-vpc-cni-init`, `kube-proxy`, and `coreDNS`.

<code>/aws/containerinsights/Cluster_Name/host</code>	Logs from <code>/var/log/dmesg</code> , <code>/var/log/secure</code> , and <code>/var/log/messages</code> .
<code>/aws/containerinsights/Cluster_Name/dataplane</code>	The logs in <code>/var/log/journal</code> for <code>kubelet.service</code> , <code>kubeproxy.service</code> , and <code>docker.service</code> .

If you don't want to use Container Insights with Fluent Bit or Fluentd for logging, you can capture node and container logs with the CloudWatch agent installed on Amazon EKS nodes. Amazon EKS nodes are EC2 instances, which means you should include them in your standard system-level logging approach for Amazon EC2. If you install the CloudWatch agent using Distributor and State Manager, then Amazon EKS nodes are also included in the CloudWatch agent installation, configuration, and update.

The following table shows logs that are specific to Kubernetes and that you must capture if you aren't using Container Insights with Fluent Bit or Fluentd for logging.

<code>/var/log/containers</code>	This directory provides symbolic links to all the Kubernetes container logs under the <code>/var/log/pods</code> directory structure. This effectively captures your application container logs writing to <code>stdout</code> or <code>stderr</code> . This includes logs for Kubernetes system containers such as <code>aws-vpc-cni-init</code> , <code>kube-proxy</code> , and <code>coreDNS</code> . Important: This is not required if you are using Container Insights.
<code>var/log/aws-routed-eni/ipamd.log</code> <code>/var/log/aws-routed-eni/plugin.log</code>	The logs for the L-IPAM daemon can be found here

You must make sure that Amazon EKS nodes install and configure the CloudWatch agent to send appropriate system-level logs and metrics. However, the Amazon EKS optimized AMI doesn't

include the Systems Manager agent. By using [launch templates](#), you can automate the Systems Manager agent installation and a default CloudWatch configuration that captures important Amazon EKS specific logs with a startup script implemented through the user data section. Amazon EKS nodes are deployed using an Auto Scaling group as either a [managed node group](#) or as [self-managed nodes](#).

With managed node groups, you supply a [launch template](#) that includes the user data section to automate the Systems Manager agent installation and CloudWatch configuration. You can customize and use the [amazon_eks_managed_node_group_launch_config.yaml](#) AWS CloudFormation template to create a launch template that installs the Systems Manager agent, CloudWatch agent, and also adds an Amazon EKS specific logging configuration to the CloudWatch configuration directory. This template can be used to update your Amazon EKS managed node groups launch template with an infrastructure-as-code (IaC) approach. Each update to the AWS CloudFormation template provisions a new version of the launch template. You can then update the node group to use the new template version and have the [managed lifecycle process](#) update your nodes without downtime. Make sure that the IAM role and instance profile applied to your managed node group includes the `CloudWatchAgentServerPolicy` and `AmazonSSMManagedInstanceCore` AWS managed policies.

With self-managed nodes, you directly provision and manage the lifecycle and update strategy for your Amazon EKS nodes. Self-managed nodes allow you to run Windows nodes on your Amazon EKS cluster and [Bottlerocket](#), along with [other options](#). You can use AWS CloudFormation to deploy self-managed nodes into your Amazon EKS clusters, which means you can use an IaC and managed change approach for your Amazon EKS clusters. AWS provides the [amazon-eks-nodegroup.yaml](#) AWS CloudFormation template that you can use as-is or customize. The template provisions all required resources for Amazon EKS nodes in a cluster (for example, a separate IAM role, security group, Amazon EC2 Auto Scaling group, and a launch template). The [amazon-eks-nodegroup.yaml](#) AWS CloudFormation template is an updated version that installs the required Systems Manager agent, CloudWatch agent, and also adds an Amazon EKS specific logging configuration to the CloudWatch configuration directory.

Logging for Amazon EKS on Fargate

With Amazon EKS on Fargate, you can deploy pods without allocating or managing your Kubernetes nodes. This removes the need to capture system-level logs for your Kubernetes nodes. To capture the logs from your Fargate pods, you can use Fluent Bit to forward the logs directly to CloudWatch. This enables you to automatically route logs to CloudWatch without further configuration or a sidecar container for your Amazon EKS pods on Fargate. For more information

about this, see [Fargate logging](#) in the Amazon EKS documentation and [Fluent Bit for Amazon EKS](#) on the AWS Blog. This solution captures the STDOUT and STDERR input/output (I/O) streams from your container and sends them to CloudWatch through Fluent Bit, based on the Fluent Bit configuration established for the Amazon EKS cluster on Fargate.

Metrics for Amazon EKS and Kubernetes

Kubernetes provides a metrics API that allows you to access resource usage metrics (for example, CPU and memory usage for nodes and pods), but the API only provides point-in-time information and not historical metrics. The [Kubernetes metrics-server](#) is typically used for Amazon EKS and Kubernetes deployments to aggregate metrics, provide short-term historical information on metrics, and support features such as [Horizontal Pod Autoscaler](#).

Amazon EKS exposes control plane metrics through the Kubernetes API server [in a Prometheus format](#) and CloudWatch can capture and ingest these metrics. CloudWatch and Container Insights can also be configured to provide comprehensive metrics capture, analysis, and alarming for your Amazon EKS nodes and pods.

Kubernetes control plane metrics

Kubernetes exposes control plane metrics in a Prometheus format by using the `/metrics` HTTP API endpoint. You should install [Prometheus](#) in your Kubernetes cluster to graph and view these metrics with a web browser. You can also [ingest the metrics exposed](#) by the Kubernetes API server into CloudWatch.

Node and system metrics for Kubernetes

Kubernetes provides the Prometheus [metrics-server](#) pod that you can [deploy and run](#) on your Kubernetes clusters for cluster, node, and pod-level CPU and memory statistics. These metrics are used with the [Horizontal Pod Autoscaler](#) and [Vertical Pod Autoscaler](#). CloudWatch can also provide these metrics.

You should install the Kubernetes Metrics Server if you use the [Kubernetes Dashboard](#) or the horizontal and vertical pod autoscalers. The Kubernetes Dashboard helps you browse and configure your Kubernetes cluster, nodes, pods and related configuration, and view the CPU and memory metrics from the Kubernetes Metrics Server. You can deploy this solution for individual clusters by following the steps from the [Deploy the Kubernetes Dashboard](#) in the Amazon EKS documentation.

The metrics provided by the Kubernetes Metrics Server can't be used for non-auto scaling purposes (for example, monitoring). The metrics are meant for point-in-time analysis and not historical analysis. The Kubernetes Dashboard deploys the `dashboard-metrics-scraper` to store metrics from the Kubernetes Metrics Server for a short time window.

Container Insights uses a containerized version of the CloudWatch agent that runs in a Kubernetes DaemonSet to discover all running containers in a cluster and provide node-level metrics. It collects performance data at every layer of the performance stack. You can use the Quick Start from AWS Quick Starts or configure Container Insights separately. The Quick Start sets up metrics monitoring with the CloudWatch agent and logging with Fluent Bit so you only need to deploy it once for logging and monitoring.

Because Amazon EKS nodes are EC2 instances, you should capture systems-level metrics, in addition to metrics captured by Container Insights, by using the standards you defined for Amazon EC2. You can use the same approach from the [Set up State Manager and Distributor for CloudWatch agent deployment and configuration](#) section of this guide to install and configure the CloudWatch agent for your Amazon EKS clusters. You can update your Amazon EKS specific CloudWatch configuration file to include metrics as well as your Amazon EKS specific log configuration.

The CloudWatch agent with Prometheus support can automatically discover and scrape the Prometheus metrics from [supported, containerized workloads and systems](#). It ingests them as CloudWatch logs in embedded metric format for analysis with CloudWatch Logs Insights and automatically creates CloudWatch metrics.

Important

You must [deploy a specialized version](#) of the CloudWatch agent to collect Prometheus metrics. This is a separate agent from the CloudWatch agent deployed for Container Insights. You can use the [prometheus_jmx](#) sample Java application, which includes the deployment and configuration files for the CloudWatch agent and Amazon EKS pod deployment to demonstrate Prometheus metrics discovery. For more information, see [Set up Java/JMX sample workload on Amazon EKS and Kubernetes](#) in the CloudWatch documentation. You can also configure the CloudWatch agent to capture metrics from other Prometheus targets running in your Amazon EKS cluster.

Application metrics

You can create your own custom metrics with the [CloudWatch embedded metric format](#). To ingest embedded metric format statements, you need to send embedded metric format entries to an embedded metric format endpoint. The CloudWatch agent can be configured as a [sidecar container in your Amazon EKS pod](#). The CloudWatch agent configuration is stored as a Kubernetes ConfigMap and read by your CloudWatch agent sidecar container to start the embedded metric format endpoint.

You can also set up your application as a Prometheus target and configure the CloudWatch agent, with Prometheus support, to discover, scrape, and ingest your metrics into CloudWatch. For example, you can use the [open-source JMX exporter](#) with your Java applications to expose JMX Beans for Prometheus consumption by the CloudWatch agent.

If you don't want to use the embedded metric format, you can also create and update CloudWatch metrics by using [AWS API](#) or AWS [SDK](#). However, we don't recommend this approach because it mixes monitoring and the application logic.

Metrics for Amazon EKS on Fargate

Fargate automatically provisions Amazon EKS nodes to run your Kubernetes pods so you don't need to monitor and collect node-level metrics. However, you must monitor metrics for pods running on your Amazon EKS nodes on Fargate. Container Insights isn't currently available for Amazon EKS on Fargate because it requires the following capabilities that aren't currently supported:

- DaemonSets aren't currently supported. Container Insights is deployed by running the CloudWatch agent as a DaemonSet on each cluster node.
- HostPath persistent volumes aren't supported. The CloudWatch agent container uses hostPath persistent volumes as a prerequisite for gathering container metric data.
- Fargate prevents privileged containers and access to host information.

You can use the [built-in log router for Fargate](#) to send embedded metric format statements to CloudWatch. The log router uses Fluent Bit, which has a CloudWatch plugin that can be configured to support embedded metric format statements.

You can retrieve and capture pod-level metrics for your Fargate nodes by deploying the Prometheus server in your Amazon EKS cluster to gather metrics from your Fargate nodes. Because

Prometheus requires persistent storage, you can deploy Prometheus on Fargate if you use Amazon Elastic File System (Amazon EFS) for persistent storage. You can also deploy Prometheus on an Amazon EC2 backed node. For more information, see [Monitoring Amazon EKS on AWS Fargate using Prometheus and Grafana](#) on the AWS Blog.

Prometheus monitoring on Amazon EKS

[Amazon Managed Service for Prometheus](#) provides a scalable, secure, AWS managed service for open-source Prometheus. You can use Prometheus query language (PromQL) to monitor the performance of containerized workloads without managing the underlying infrastructure for ingesting, storing, and querying operational metrics. You can collect Prometheus metrics from Amazon EKS and Amazon ECS by using [AWS Distro for OpenTelemetry \(ADOT\)](#) or Prometheus servers as collection agents.

[CloudWatch Container Insights monitoring for Prometheus](#) enables you to configure and use the CloudWatch agent to discover Prometheus metrics from Amazon ECS, Amazon EKS, and Kubernetes workloads, and ingest them as CloudWatch metrics. This solution is appropriate if CloudWatch is your primary observability and monitoring solution. However, the following list outlines use cases where Amazon Managed Service for Prometheus provides more flexibility for ingesting, storing, and querying Prometheus metrics:

- Amazon Managed Service for Prometheus enables you to use existing Prometheus servers deployed in Amazon EKS or self-managed Kubernetes and configure them to write to Amazon Managed Service for Prometheus instead of a locally configured data store. This removes the undifferentiated heavy lifting of managing a highly available data store for your Prometheus servers and its infrastructure. Amazon Managed Service for Prometheus is a suitable choice when you have a mature Prometheus deployment that you want to leverage in the AWS Cloud.
- Grafana directly supports Prometheus as a data source for visualization. If you want to use Grafana with Prometheus instead of CloudWatch Dashboards for your container monitoring, then Amazon Managed Service for Prometheus could meet your requirements. Amazon Managed Service for Prometheus integrates with Amazon Managed Grafana to provide a managed open-source monitoring and visualization solution.
- Prometheus enables you to perform analysis on your operational metrics by using PromQL queries. In contrast, [the CloudWatch agent ingests Prometheus metrics in embedded metric format](#) into CloudWatch Logs which result in CloudWatch metrics. You can query the embedded metric format logs by using CloudWatch Logs Insights.
- If you don't plan to use CloudWatch for monitoring and metrics capture, then you should use Amazon Managed Service for Prometheus with your Prometheus server and a visualization solution such as Grafana. You need to configure your Prometheus server to scrape metrics from your Prometheus targets and configure the server to [remote write to your Amazon Managed Service for Prometheus workspace](#). If you use Amazon Managed Grafana, then you can [directly](#)

[integrate Amazon Managed Grafana with your Amazon Managed Service for Prometheus data source by using the included plugin](#). Because metric data is stored in Amazon Managed Service for Prometheus, there is no dependency to deploy the CloudWatch agent or requirement to ingest data into CloudWatch. The CloudWatch agent is required for Container Insights monitoring for Prometheus.

You can also use the ADOT Collector to scrape from a Prometheus-instrumented application and send the metrics to Amazon Managed Service for Prometheus. For more information about ADOT Collector, see the [AWS Distro for OpenTelemetry](#) documentation.

Logging and metrics for AWS Lambda

[Lambda](#) removes the need to manage and monitor servers for your workloads and automatically works with CloudWatch Metrics and CloudWatch Logs without further configuration or instrumentation of your application's code. This section helps you understand the performance characteristics of the systems used by Lambda and how your configuration choices influence performance. It also helps you log and monitor your Lambda functions for performance optimization and diagnosing application-level issues.

Lambda function logging

Lambda automatically streams standard output and standard error messages from a Lambda function to CloudWatch Logs, without requiring logging drivers. Lambda also automatically provisions containers that run your Lambda function and configures them to output log messages in separate log streams.

Subsequent invocations of your Lambda function can reuse the same container and output to the same log stream. Lambda can also provision a new container and output the invocation to a new log stream.

Lambda automatically creates a log group when your Lambda function is first invoked. Lambda functions can have multiple versions and you can choose the version that you want to run. All logs for the Lambda function's invocations are stored in the same log group. The name cannot be changed and is in the `/aws/lambda/<YourLambdaFunctionName>` format. A separate log stream is created in the log group for each Lambda function instance. Lambda has a standard naming convention for log streams that uses a `YYYY/MM/DD/[<FunctionVersion>]<InstanceId>` format. The `InstanceId` is generated by AWS to identify the Lambda function instance.

We recommend that you format your log messages in JSON format because you can query them more easily with CloudWatch Logs Insights. They can also be more easily filtered and exported. You can use a logging library to simplify this process or write your own log handling functions. We recommend that you use a logging library to help format and classify log messages. For example, if your Lambda function is written in Python, you can use the [Python logging module](#) to log messages and control the output format. Lambda natively uses the Python logging library for Lambda functions written in Python, and you can retrieve and customize the logger within your

Lambda function. AWS Labs has created the [AWS Lambda Powertools for Python](#) developer toolkit to make it easier to enrich log messages with key data such as cold starts. The toolkit is available for Python, Java, Typescript, and .NET.

Another best practice is to set the log output level by using a variable and adjust it based on the environment and your requirements. Your Lambda function's code, in addition to the libraries used, could output a large amount of log data depending on the log output level. This can impact your logging costs and affect performance.

Lambda allows you to set environment variables for your Lambda function runtime environment without updating your code. For example, you can create a `LAMBDA_LOG_LEVEL` environment variable that defines the log output level that you can retrieve from your code. The following example attempts to retrieve a `LAMBDA_LOG_LEVEL` environment variable and use the value to define the logging output. If the environment variable is not set, it defaults to the `INFO` level.

```
import logging
from os import getenv

logger = logging.getLogger()
log_level = getenv("LAMBDA_LOG_LEVEL", "INFO")
level = logging.getLevelName(log_level)
logger.setLevel(level)
```

Sending logs to other destinations from CloudWatch

You can send logs to other destinations (for example, Amazon OpenSearch Service or a Lambda function) by using subscription filters. If you don't use Amazon OpenSearch Service, you can use a Lambda function to process the logs and send them to an AWS service of your choice using the AWS SDKs.

You can also use SDKs for log destinations outside the AWS Cloud in your Lambda function to directly send log statements to a destination of your choice. If you choose this option, we recommend that you consider the impact of the latency, additional processing time, error and retry handling, and coupling of operational logic to your Lambda function.

Lambda function metrics

Lambda lets you run your code without managing or scaling servers and this almost removes the burden of system-level auditing and diagnostics. However, it's still important to understand performance and invocation metrics at the system level for your Lambda functions. This helps you optimize the resource configuration and improve code performance. Effectively monitoring and measuring performance can improve user experience and lower your costs by appropriately sizing your Lambda functions. Typically, workloads running as Lambda functions also have application-level metrics that need to be captured and analyzed. Lambda directly supports the embedded metric format to make capturing application-level CloudWatch metrics easier.

System-level metrics

Lambda automatically integrates with CloudWatch Metrics and provides a set of [standard metrics for your Lambda functions](#). Lambda also provides a separate monitoring dashboard for each Lambda function with these metrics. Two important metrics that you need to monitor are errors and invocation errors. Understanding the differences between invocation errors and other error types helps you diagnose and support Lambda deployments.

[Invocation errors](#) prevent your Lambda function from running. These errors occur before your code is run so you can't implement error handling within your code to identify them. Instead, you should configure alarms for your Lambda functions that detect these errors and notify the operations and workload owners. These errors are often related to a configuration or permission error and can occur because of a change in your configuration or permissions. Invocation errors might initiate a retry, which causes multiple invocations of your function.

A successfully invoked Lambda function returns an HTTP 200 response even if an exception is thrown by the function. Your Lambda functions should implement error handling and raise exceptions so that the `Errors` metric captures and identifies failed runs of your Lambda function. You should return a formatted response from your Lambda function invocations that includes information to determine whether the run failed completely, partially, or was successful.

CloudWatch provides [CloudWatch Lambda Insights](#) that you can enable for individual Lambda function. Lambda Insights collects, aggregates, and summarizes system-level metrics (for example, CPU time, memory, disk and network usage). Lambda Insights also collects, aggregates, and summarizes diagnostic information (for example, cold starts and Lambda worker shutdowns) to help you isolate and quickly resolve issues.

Lambda Insights uses the embedded metric format to automatically emit performance information to the `/aws/lambda-insights/` log group with a log stream name prefix based on your Lambda function's name. These performance log events create CloudWatch metrics that are the basis for automatic CloudWatch dashboards. We recommend that you enable Lambda Insights for performance testing and production environments. Additional metrics created by Lambda Insights include `memory_utilization` that helps correctly size Lambda functions so that you avoid paying for unrequired capacity.

Application metrics

You can also create and capture your own application metrics in CloudWatch using the embedded metric format. You can leverage [AWS provided libraries for embedded metric format](#) to create and emit embedded metric format statements to CloudWatch. The integrated Lambda CloudWatch logging facility is configured to process and extract appropriately formatted embedded metric format statements.

Searching and analyzing logs in CloudWatch

After your logs and metrics are captured into a consistent format and location, you can search and analyze them to help improve operational efficiency, in addition to identifying and troubleshooting issues. We recommend that you capture your logs in a well-formed format (for example, JSON) to make it easier to search and analyze your logs. Most workloads use a collection of AWS resources such as network, compute, storage, and databases. Where possible, you should collectively analyze the metrics and logs from these resources and correlate them in order to effectively monitor and manage all of your AWS workloads.

CloudWatch provides several features to help analyze logs and metrics, such as [CloudWatch Application Insights](#) to collectively define and monitor metrics and logs for an application across different AWS resources, [CloudWatch Anomaly Detection](#) to surface anomalies for your metrics, and [CloudWatch Log Insights](#) to interactively search and analyze your log data in CloudWatch Logs.

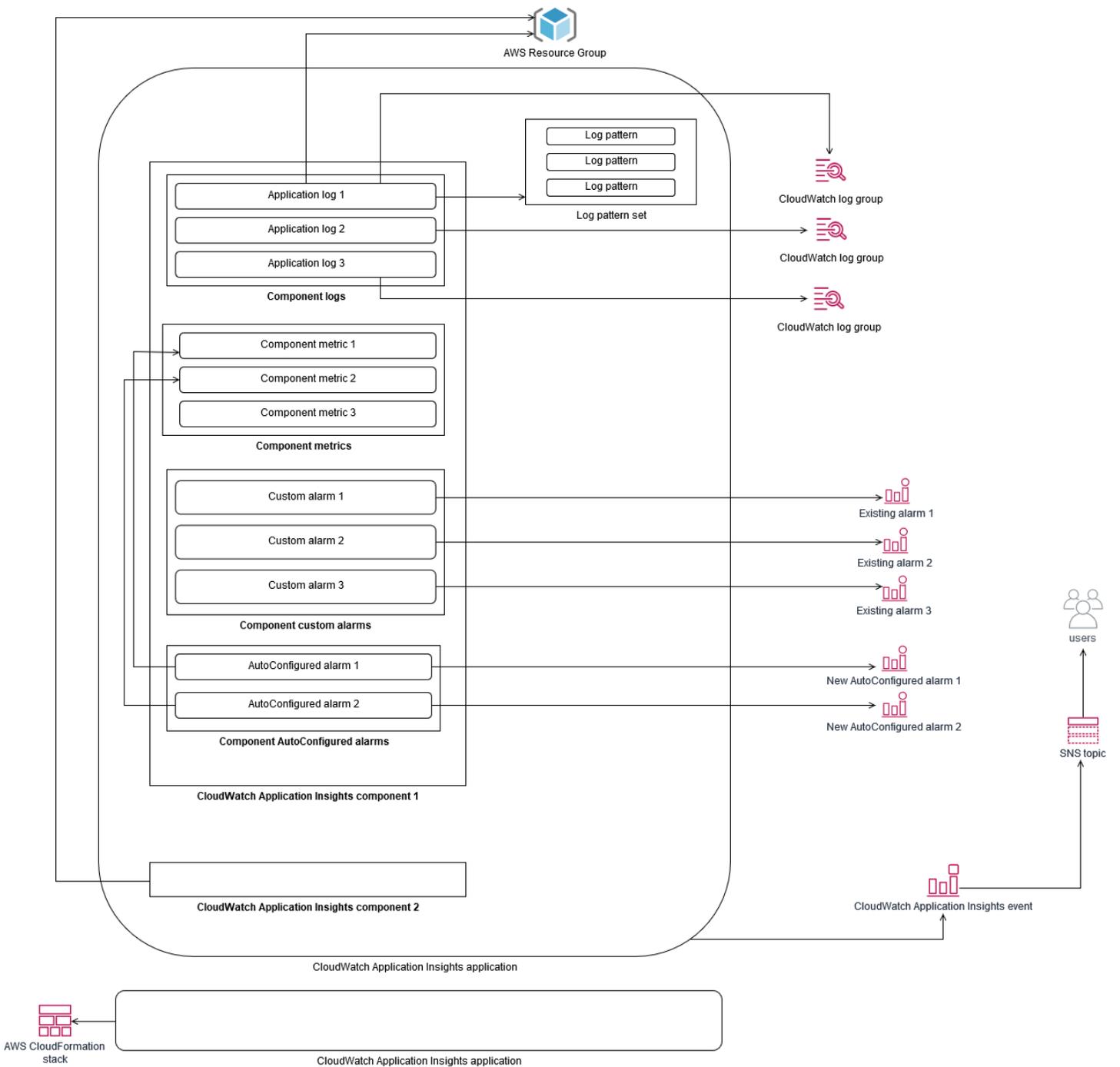
Collectively monitor and analyze applications with CloudWatch Application Insights

Application owners can use Amazon CloudWatch Application Insights to set up automatic monitoring and analysis for workloads. This can be configured in addition to standard systems-level monitoring configured for all workloads in an account. Setting up monitoring through CloudWatch Application Insights can also help application teams proactively align to operations and reduce mean time to recovery (MTTR). CloudWatch Application Insights can help reduce the effort required to establish application-level logging and monitoring. It also provides a component-based framework that assists teams in dividing logging and monitoring responsibilities.

CloudWatch Application Insights uses resource groups to identify the resources that should be collectively monitored as an application. The supported resources in the resource group become individually defined components of your CloudWatch Application Insights application. Each component of your CloudWatch Application Insights application has its own logs, metrics, and alarms.

For logs, you define the log pattern set that should be used for the component and within your CloudWatch Application Insights application. A log pattern set is a collection of log patterns to search for based on regular expressions, along with a low, medium, or high severity for when the pattern is detected. For metrics, you choose the metrics to monitor for each component from

a list of service-specific and supported metrics. For alarms, CloudWatch Application Insights automatically creates and configures standard or anomaly detection alarms for the metrics being monitored. CloudWatch Application Insights has automatic configurations for metrics and log capture for the technologies outlined in the [Logs and metrics supported by CloudWatch Application Insights](#) in the CloudWatch documentation. The following diagram shows the relationships between CloudWatch Application Insights components and their logging and monitoring configurations. Each component has defined its own logs and metrics to monitor using CloudWatch logs and metrics.



EC2 instances monitored by CloudWatch Application Insights require Systems Manager and CloudWatch agents and permissions. For more information about this, see [Prerequisites to configure an application with CloudWatch Application Insights](#) in the CloudWatch documentation. CloudWatch Application Insights uses Systems Manager to install and update the CloudWatch agent. The metrics and logs configured in CloudWatch Application Insights create a CloudWatch agent configuration file that is stored in a Systems Manager parameter with the

AmazonCloudWatch-ApplicationInsights-SSMParameter prefix for each CloudWatch Application Insights component. This results in a separate CloudWatch agent configuration file being added to the CloudWatch agent configuration directory on the EC2 instance. A Systems Manager command is run to append this configuration to the active configuration on the EC2 instance. Using CloudWatch Application Insights doesn't impact existing CloudWatch agent configuration settings. You can use CloudWatch Application Insights in addition to your own system and application-level CloudWatch agent configurations. However, you should ensure that the configurations don't overlap.

Performing log analysis with CloudWatch Logs Insights

CloudWatch Logs Insights makes it easy to search multiple log groups by using a simple query language. If your application logs are structured in JSON format, CloudWatch Logs Insights automatically discovers the JSON fields across your log streams in multiple log groups. You can use CloudWatch Logs Insights to analyze your application and system logs, which saves your queries for future use. The query syntax for CloudWatch Logs Insights supports functions such as aggregation with functions, for example, `sum()`, `avg()`, `count()`, `min()`, and `max()`, that can be helpful for troubleshooting your applications or performance analysis.

If you use the embedded metric format to create CloudWatch metrics, you can query your embedded metric format logs to generate one-time metrics by using the supported aggregation functions. This helps reduce your CloudWatch monitoring costs by capturing data points necessary to generate specific metrics on an as-needed basis, instead of actively capturing them as custom metrics. This is especially effective for dimensions with high cardinality that would result in a large number of metrics. CloudWatch Container Insights also takes this approach and captures detailed performance data but only generates CloudWatch metrics for a subset of this data.

For example, the following embedded metric entry only generates a limited set of CloudWatch metrics from the metric data that is captured in the embedded metric format statement:

```
{
  "AutoScalingGroupName": "eks-e0bab7f4-fa6c-64ba-dbd9-094aee6cf9ba",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "pod_number_of_container_restarts"
        }
      ]
    }
  ]
}
```

```
],
"Dimensions": [
  [
    "PodName",
    "Namespace",
    "ClusterName"
  ]
],
"Namespace": "ContainerInsights"
}
],
"ClusterName": "eksdemo",
"InstanceId": "i-03b21a16b854aa4ca",
"InstanceType": "t3.medium",
"Namespace": "amazon-cloudwatch",
"NodeName": "ip-172-31-10-211.ec2.internal",
"PodName": "cloudwatch-agent",
"Sources": [
  "cadvisor",
  "pod",
  "calculated"
],
"Timestamp": "1605111338968",
"Type": "Pod",
"Version": "0",
"pod_cpu_limit": 200,
"pod_cpu_request": 200,
"pod_cpu_reserved_capacity": 10,
"pod_cpu_usage_system": 3.268605094109382,
"pod_cpu_usage_total": 8.899539221131045,
"pod_cpu_usage_user": 4.160042847048305,
"pod_cpu_utilization": 0.44497696105655227,
"pod_cpu_utilization_over_pod_limit": 4.4497696105655224,
"pod_memory_cache": 4096,
"pod_memory_failcnt": 0,
"pod_memory_hierarchical_pgfault": 0,
"pod_memory_hierarchical_pgmajfault": 0,
"pod_memory_limit": 209715200,
"pod_memory_mapped_file": 0,
"pod_memory_max_usage": 43024384,
"pod_memory_pgfault": 0,
"pod_memory_pgmajfault": 0,
"pod_memory_request": 209715200,
```

```
"pod_memory_reserved_capacity": 5.148439982463127,  
"pod_memory_rss": 38481920,  
"pod_memory_swap": 0,  
"pod_memory_usage": 42803200,  
"pod_memory_utilization": 0.6172094650851303,  
"pod_memory_utilization_over_pod_limit": 11.98828125,  
"pod_memory_working_set": 25141248,  
"pod_network_rx_bytes": 3566.4174629544723,  
"pod_network_rx_dropped": 0,  
"pod_network_rx_errors": 0,  
"pod_network_rx_packets": 3.3495665260575094,  
"pod_network_total_bytes": 4283.442421354973,  
"pod_network_tx_bytes": 717.0249584005006,  
"pod_network_tx_dropped": 0,  
"pod_network_tx_errors": 0,  
"pod_network_tx_packets": 2.6964010534762948,  
"pod_number_of_container_restarts": 0,  
"pod_number_of_containers": 1,  
"pod_number_of_running_containers": 1,  
"pod_status": "Running"  
}
```

However, you can query the captured metrics to gain further insights. For example, you can run the following query to see the latest 20 pods with memory page faults:

```
fields @timestamp, @message  
| filter (pod_memory_pgfault > 0)  
| sort @timestamp desc  
| limit 20
```

Performing log analysis with Amazon OpenSearch Service

CloudWatch integrates with [Amazon OpenSearch Service](#) by enabling you to stream log data from CloudWatch log groups to an Amazon OpenSearch Service cluster of your choice with a [subscription filter](#). You can use CloudWatch for primary log and metrics capture and analysis, and then augment it with Amazon OpenSearch Service for the following use cases:

- **Fine-grained data access control** – Amazon OpenSearch Service enables you to limit access to data down to the field level and helps anonymize data in fields based on user permissions. This is useful if you want support troubleshooting without exposing sensitive data.

- **Aggregate and search logs across multiple accounts, Regions, and infrastructure** – You can stream your logs from multiple accounts and Regions into a common Amazon OpenSearch Service cluster. Your centralized operations teams can analyze trends, issues, and perform analytics across accounts and Regions. Streaming CloudWatch logs to Amazon OpenSearch Service also helps you search and analyze a multi-Region application in a central location.
- **Ship and enrich logs directly to Amazon OpenSearch Service by using ElasticSearch agents** – Your application and technology stack components can use OSs that are not supported by the CloudWatch agent. You might also want to enrich and transform log data before it is shipped to your logging solution. Amazon OpenSearch Service supports standard Elasticsearch clients such as the [Elastic Beats family data shippers](#) and [Logstash](#) that support log enrichment and transformation before sending the log data to Amazon OpenSearch Service.
- **Existing operations management solution uses an [ElasticSearch, Logstash, Kibana \(ELK\) Stack for logging and monitoring](#)** – You might already have a significant investment in Amazon OpenSearch Service or open-source Elasticsearch with many workloads already configured. You might also have operational dashboards that have been created in [Kibana](#) that you want to continue to use.

If you don't plan to use CloudWatch logs, you can use Amazon OpenSearch Service supported agents, log drivers, and libraries (for example, Fluent Bit, Fluentd, [logstash](#), and the [Open Distro for ElasticSearch API](#)) to ship your logs directly to Amazon OpenSearch Service and bypass CloudWatch. However, you should also implement a solution to capture logs generated by AWS services. CloudWatch Logs is the primary log capture solution for many AWS services and multiple services automatically create new log groups in CloudWatch. For example, Lambda creates a new log group for every Lambda function. You can set up a subscription filter for a log group to stream its logs to Amazon OpenSearch Service. You can manually configure a subscription filter for each individual log group that you want to stream to Amazon OpenSearch Service. Alternatively, you can deploy a solution that automatically subscribes new log groups to Elasticsearch clusters. You can stream logs to an Elasticsearch cluster in the same account or a centralized account. Streaming logs to an Elasticsearch cluster in the same account helps workload owners to better analyze and support their workloads.

You should consider setting up an Elasticsearch cluster in a centralized or shared account for aggregating logs across your accounts, Regions, and applications. For example, AWS Control Tower sets up a Log Archive account that is used for centralized logging. When a new account is created in AWS Control Tower, its AWS CloudTrail and AWS Config logs are delivered to an S3 bucket in this

centralized account. The logging instrumented by AWS Control Tower is for configuration, change, and audit logging.

To establish a centralized application log analysis solution with Amazon OpenSearch Service, you can deploy one or more centralized Amazon OpenSearch Service clusters to your centralized logging account and configure log groups in your other accounts to stream logs to the centralized Amazon OpenSearch Service clusters.

You can create separate Amazon OpenSearch Service clusters to handle different applications or layers of your cloud architecture that might be distributed across your accounts. Using separate Amazon OpenSearch Service clusters helps you reduce your security and availability risk and having a common Amazon OpenSearch Service cluster can make it easier to search and relate data within the same cluster.

Alarming options with CloudWatch

Performing one-time and automated analysis of important metrics helps you detect and resolve issues before they impact your workloads. CloudWatch makes it easy to graph and compare multiple metrics by using multiple statistics over a specific time period. You can use CloudWatch to search across all metrics with the required dimension values to find the metrics that you need for your analysis.

We recommend that you begin your metrics capture approach by including an initial set of metrics and dimensions to use as a baseline for monitoring a workload. Over time, the workload matures and you can add additional metrics and dimensions to help you further analyze and support it. Your applications or workloads might use multiple AWS resources and have their own custom metrics, you should group these resources under a namespace to make them easier to identify.

You should also consider how logging and monitoring data is correlated so that you can quickly identify the relevant logging and monitoring data to diagnose specific issues. You can use [CloudWatch ServiceLens](#) to correlate traces, metrics, logs, and alarms for diagnosing issues. You should also consider including additional dimensions in metrics and identifiers in logs for your workloads to help you quickly search for and identify issues across systems and services.

Using CloudWatch alarms to monitor and alarm

You can use [CloudWatch alarms](#) to reduce manual monitoring in your workloads or applications. You should begin by reviewing the metrics that you are capturing for each workload component and determine the appropriate thresholds for each metric. Make sure that you identify which team members must be notified when a threshold is breached. You should establish and target distribution groups, rather than individual team members.

CloudWatch alarms can integrate with your service management solution to automatically create new tickets and run operational workflows. For example, AWS provides the AWS Service Management Connector for [ServiceNow](#) and [Jira Service Desk](#) to help you quickly set up integrations. This approach is critical to ensuring that raised alarms are acknowledged and aligned to your existing operations workflows that might already be defined in these products.

You can also create multiple alarms for the same metric that have different thresholds and evaluation periods, which helps establish an escalation process. For example, if you have a

`OrderQueueDepth` metric that tracks customer orders, you might define a lower threshold over a short one-minute average period that notifies application team members by email or [Slack](#). You can also define another alarm for the same metric over a longer 15-minute period at the same threshold and that pages, emails, and notifies the application team and application team's lead. Finally, you can define a third alarm for a hard average threshold over a 30-minute period that notifies upper-management and notifies all team members previously notified. Creating multiple alarms helps you take different actions for different conditions. You can begin with a simple notification process and then adjust and improve it as required.

Using CloudWatch anomaly detection to monitor and alarm

You can use [CloudWatch anomaly detection](#) if you are unsure about the thresholds to apply for a particular metric or if you want an alarm to automatically adjust the threshold values based on observed, historical values. CloudWatch anomaly detection is particularly useful for metrics that might have regular, predictable changes in activity, for example, daily purchase orders for same-day delivery increasing before a cutoff time. Anomaly detection enables thresholds that adjust automatically and can help reduce false alarms. You can enable anomaly detection for each metric and statistic, and configure CloudWatch to alarm based on outliers.

For example, you can enable anomaly detection for the `CPUUtilization` metric and the `AVG` statistic on an EC2 instance. Anomaly detection then uses up to 14 days of historical data to create the machine learning (ML) model. You can create multiple alarms with different anomaly detection bands to establish an alarm escalation process, similar to creating multiple standard alarms with different thresholds.

For more information about this section, see [Creating a CloudWatch alarm based on anomaly detection](#) in the CloudWatch documentation.

Alarming across multiple Regions and accounts

Application and workload owners should create application-level alarms for workloads that span multiple Regions. We recommend creating separate alarms within each account and Region that your workload is deployed in. You can simplify and automate this process by using account and Region agnostic AWS CloudFormation StackSets and templates to deploy application resources with the required alarms. You can configure the alarm actions to target a common Amazon Simple Notification Service (Amazon SNS) topic, which means the same notification or remediation action is used regardless of the account or Region.

In multi-account and multi-Region environments, we recommend that you create aggregated alarms for your accounts and Regions to monitor account and Regional issues by using AWS CloudFormation StackSets and aggregate metrics, such as average CPUUtilization across all EC2 instances.

You should also consider creating standard alarms for each workload that is configured for the standard CloudWatch metrics and logs that you capture. For example, you can create a separate alarm for each EC2 instance that monitors the CPU utilization metric and notifies a central operations team when average CPU utilization is over 80% on a daily basis. You can also create a standard alarm that monitors average CPU utilization under 10% on a daily basis. These alarms help the central operations team to work with specific workload owners to change the size of the EC2 instances when required.

Automating alarm creation with EC2 instance tags

Creating a standard set of alarms for your EC2 instances can be time consuming, inconsistent, and error prone. You can accelerate the alarm creation process by using the [amazon-cloudwatch-auto-alarms](#) solution to automatically create a standard set of CloudWatch alarms for your EC2 instances and create custom alarms based on EC2 instance tags. The solution removes the need to manually create standard alarms and can be useful during a large-scale migration of EC2 instances that uses tools such as CloudEndure. You can also deploy this solution with AWS CloudFormation StackSets to support multiple Regions and accounts. For more information, see [Use tags to create and maintain Amazon CloudWatch alarms for Amazon EC2 instances](#) on the AWS Blog.

Monitoring application and service availability

CloudWatch helps you monitor and analyze the performance and runtime aspects of your applications and workloads. You should also monitor the availability and reachability aspects of your applications and workloads. You can achieve this by using an active monitoring approach with [Amazon Route 53 health checks](#) and [CloudWatch Synthetics](#).

You can use Route 53 health checks when you want to monitor connectivity to a webpage through HTTP or HTTPS, or network connectivity through TCP to a public Domain Name System (DNS) name or IP address. Route 53 health checks initiate connections from the Regions that you specify on ten-second or 30-second intervals. You can choose multiple Regions for the health check to run in, each health check runs independently, and you must choose at least three Regions. You can search the response body of an HTTP or HTTPS request for a specific substring if it appears in the first 5,120 bytes of data returned for health check evaluation. An HTTP or HTTPS request is considered healthy if it returns a 2xx or 3xx response. Route 53 health checks can be used to create a composite health check by checking the health of other health checks. You can do this if you have multiple service endpoints and you want to perform the same notification when one of them becomes unhealthy. If you use Route 53 for DNS, you can configure Route 53 to [fail over to another DNS entry](#) if a health check becomes unhealthy. For each critical workload, you should consider setting up Route 53 health checks for external endpoints that are critical for normal operations. Route 53 health checks can help you avoid writing failover logic into your applications.

CloudWatch synthetics allows you to define a canary as a script to evaluate the health and availability of your workloads. Canaries are scripts written in Node.js or Python and work over HTTP or HTTPS protocols. They create Lambda functions in your account that use Node.js or Python as a framework. Each canary that you define can perform multiple HTTP or HTTPS calls to different endpoints. This means you can monitor the health of a series of steps, such as a use case or an endpoint with downstream dependencies. Canaries create CloudWatch metrics that include each step that was run so you can alarm and measure different steps independently. Although canaries require more planning and effort to develop than Route 53 health checks, they provide you with a highly customizable monitoring and evaluation approach. Canaries also support private resources running within your virtual private cloud (VPC), which makes them ideal for availability monitoring when you don't have a public IP address for the endpoint. You can also use canaries to monitor on-premises workloads as long as you have connectivity from within the VPC to the endpoint. This is particularly important when you have a workload that includes endpoints that exist on premises.

Tracing applications with AWS X-Ray

A request through your application might consist of calls to databases, applications, and web services running in on-premises servers, Amazon EC2, containers, or Lambda. By implementing application tracing, you can quickly identify the root cause of issues in your applications that use distributed components and services. You can use [AWS X-Ray](#) to trace your application requests across multiple components. X-Ray samples and visualizes requests on a [service graph](#) when they flow through your application components and each component is represented as a segment. X-Ray generates trace identifiers so that you can correlate a request when it flows through multiple components, which helps you view the request from end to end. You can further enhance this by including annotations and metadata to help uniquely search for and identify the characteristics of a request.

We recommend that you configure and instrument each server or endpoint in your application with X-Ray. X-Ray is implemented in your application code by making calls to the X-Ray service. X-Ray also provides AWS SDKs for multiple languages, including instrumented clients that automatically send data to X-Ray. The X-Ray SDKs provide patches to common libraries used for making calls to other services (for example, HTTP, MySQL, PostgreSQL, or MongoDB).

X-Ray provides an X-Ray daemon that you can install and run on Amazon EC2 and Amazon ECS to relay data to X-Ray. X-Ray creates traces for your application that capture performance data from the servers and containers running the X-Ray daemon that serviced the request. X-Ray automatically instruments your calls to AWS services, such as Amazon DynamoDB, as subsegments through patching the AWS SDK. X-Ray can also automatically integrate with Lambda functions.

If your application components make calls to external services that can't configure and install the X-Ray daemon or instrument the code, you can create [subsegments to wrap calls to external services](#). X-Ray correlates CloudWatch logs and metrics with your application traces if you are using the AWS X-Ray SDK for Java, which means you can quickly analyze the related metrics and logs for requests.

Deploying the X-Ray daemon to trace applications and services on Amazon EC2

You need to install and run the X-Ray daemon on the EC2 instances that your application components or microservices run on. You can use a [user data script](#) to deploy the X-Ray daemon

when EC2 instances are provisioned or you can include it in the AMI build process if you create your own AMIs. This can be particularly useful when EC2 instances are ephemeral.

You should use State Manager to ensure that the X-Ray daemon is consistently installed on your EC2 instances. For Amazon EC2 Windows instances, you can use the Systems Manager [AWS-RunPowerShellScript document](#) to run the [Windows script](#) that downloads and installs the X-Ray agent. For EC2 instances on Linux, you can use the AWS-RunShellScript document to run the Linux script that [downloads and installs the agent as a service](#).

You can use the Systems Manager [AWS-RunRemoteScript document](#) to run the script in a multi-account environment. You must create an S3 bucket that is accessible from all your accounts and we recommend [creating an S3 bucket with an organization-based bucket policy](#) if you use AWS Organizations. You then upload the scripts to the S3 bucket but make sure that the IAM role for your EC2 instances has permission to access the bucket and scripts.

You can also configure State Manager to associate the scripts to EC2 instances that have the X-Ray agent installed. Because all of your EC2 instances might not require or use X-Ray, you can target the association with instance tags. For example, you can create the State Manager association based on the presence of `InstallAWSXRayDaemonWindows` or `InstallAWSXRayDaemonLinux` tags.

Deploying the X-Ray daemon to trace applications and services on Amazon ECS or Amazon EKS

You can deploy the [X-Ray daemon](#) as a sidecar container for container-based workloads such as Amazon ECS or Amazon EKS. Your application containers can then connect to your sidecar container with container linking if you use Amazon ECS, or the container can directly connect to the sidecar container on localhost if you use [awsvpc network mode](#).

For Amazon EKS, you can define the X-Ray daemon in your application's pod definition and then your application can connect to the daemon over localhost on the container port that you specified.

Configuring Lambda to trace requests to X-Ray

Your application might include calls to Lambda functions. You don't need to install the X-Ray daemon for Lambda because the daemon process is fully managed by Lambda and cannot be

configured by the user. You can enable it for your Lambda function by using the AWS Management Console and checking the **Active Tracing** option in the X-Ray console.

For further instrumentation, you can bundle the X-Ray SDK with your Lambda function to record outgoing calls and add annotations or metadata.

Instrumenting your applications for X-Ray

You should evaluate the X-Ray SDK that aligns with your application's programming language and classify all calls that your application makes to other systems. Review the clients provided by the library that you chose and see if the SDK can automatically instrument tracing for your application's request or response. Determine if the clients provided by the SDK can be used for other downstream systems. For external systems that your application calls and that you can't instrument with X-Ray, you should create a custom subsegments to capture and identify them in your trace information.

When you instrument your application, make sure that you create annotations to help you to identify and search for requests. For example, your application might use an identifier for customers, such as `customer_id`, or segment different users based on their role in the application.

You can create a maximum of 50 annotations for each trace but you can create a metadata object containing one or more fields as long as the segment document doesn't exceed 64 kilobytes. You should selectively use annotations to locate information and use the metadata object to provide more context that helps troubleshoot the request after it is located.

Configuring the X-Ray sampling rules

By [customizing sampling rules](#), you can control the amount of data that you record and modify the sampling behavior without modifying or redeploying your code. Sampling rules tell the X-Ray SDK how many requests to record for a set of criteria. By default, the X-Ray SDK records the first request each second and five percent of any additional requests. One request per second is the reservoir. This ensures that at least one trace is recorded each second as long as the service is serving requests. Five percent is the rate at which additional requests are sampled beyond the reservoir size.

You should review and update the default configuration to determine an appropriate value for your account. Your requirements might vary in development, test, performance test, and production environments. You might have applications that require their own sampling rules based on the

amount of traffic that they receive or their level of criticality. You should begin with a baseline and regularly re-evaluate whether the baseline meets your requirements.

Dashboards and visualizations with CloudWatch

Dashboards help you quickly focus on areas of concern for applications and workloads. CloudWatch provides automatic dashboards and you can also easily create dashboards that use CloudWatch metrics. CloudWatch dashboards provide more insight than viewing metrics in isolation because they help you correlate multiple metrics and identify trends. For example, a dashboard that includes orders received, memory, CPU utilization, and database connections can help you correlate changes in workload metrics across multiple AWS resources while your order count is increasing or decreasing.

You should create dashboards at the account and application-level to monitor workloads and applications. You can get started by using CloudWatch automatic dashboards, which are AWS service-level dashboards preconfigured with service-specific metrics. Automatic service dashboards display all the standard CloudWatch metrics for the service. The automatic dashboards graph all resources used for each service metric and help you quickly identify outlier resources across your account. This can help you identify resources with high and low utilization, which can help you optimize your costs.

Creating cross-service dashboards

You can create cross-service dashboards by viewing the automatic service-level dashboard for an AWS service and using the **Add to dashboard** option from the **Actions** menu. You can then add metrics from other automatic dashboards to your new dashboard and remove metrics to narrow the dashboard's focus. You should also add your own custom metrics to track key observations (for example, orders received or transactions per second). Creating your own custom cross-service dashboard helps you focus on the most relevant metrics for your workload. We recommend that you create account-level, cross-service dashboards that cover key metrics and display all of the workloads in an account.

If you have a central office space or common area for your cloud operations teams, you can display the CloudWatch dashboard on a large TV monitor in full screen mode with automatic refresh.

Creating application or workload-specific dashboards

We recommend that you create application and workload-specific dashboards that focus on key metrics and resources for every critical application or workload in your production environment.

Application and workload-specific dashboards focus on your custom application or workload metrics and important AWS resource metrics that influence their performance.

You should regularly evaluate and customize your CloudWatch application or workload dashboards to track key metrics after incidents occur. You should also update application or workload-specific dashboards when features are introduced or retired. Updates to workload and application-specific dashboards should be a required activity for continuous improvement in quality, in addition to logging and monitoring.

Creating cross-account or cross-Region dashboards

AWS resources are primarily Regional and the metrics, alarms, and dashboards are specific to the Region that the resources are deployed in. This can require you to change Regions to view metrics, dashboards, and alarms for cross-Region workloads and applications. If you separate your applications and workloads into multiple accounts, you might also be required to re-authenticate and sign in to each account. However, CloudWatch supports cross-account and cross-Region data viewing from a single account, which means that you can view metrics, alarms, dashboards, and log widgets in a single account and Region. This is very useful if you have a centralized logging and monitoring account.

Account owners and application team owners should create dashboards for account-specific, cross-Region applications to effectively monitor key metrics in a centralized location. CloudWatch dashboards automatically support cross-Region widgets, which means you can create a dashboard that includes metrics from multiple Regions without further configuration.

An important exception is the CloudWatch Logs Insights widget because log data can only be displayed for the account and Region that you are currently logged into. You can create Region-specific metrics from your logs by using metric filters and these metrics can be displayed on a cross-Region dashboard. You can then switch to the specific Region when you need to further analyze those logs.

Operations teams should create centralized dashboard that monitor important cross-account and cross-Region metrics. For example, you can create a cross-account dashboard that includes the aggregate CPU utilization in each account and Region. You can also use [metric math](#) to aggregate and dashboard data across multiple accounts and Regions.

Using metric math to fine-tune observability and alarming

You can use metric math to help calculate metrics in formats and expressions that are relevant for your workloads. The calculated metrics can be saved and viewed on a dashboard for tracking purposes. For example, standard Amazon EBS volume metrics provide the number of read (`VolumeReadOps`) and write (`VolumeWriteOps`) operations performed over a specific period.

However, AWS provides guidelines on Amazon EBS volume performance in IOPS. You can graph and calculate the IOPS for your Amazon EBS volume in metric math by adding the `VolumeReadOps` and `VolumeWriteOps` and then dividing by the period chosen for these metrics.

In this example, we sum up the IOPS in the period and then divide by the period length to get the IOPS. You can then set an alarm against this metric math expression to alert you when your volume's IOPS approaches maximum capacity for its volume type. For more information and examples about using metric math to monitor Amazon Elastic File System (Amazon EFS) file systems with CloudWatch metrics, see [Amazon CloudWatch metric math simplifies near real-time monitoring of your Amazon EFS file systems and more](#) on the AWS Blog.

Using automatic dashboards for Amazon ECS, Amazon EKS, and Lambda with CloudWatch Container Insights and CloudWatch Lambda Insights

CloudWatch Container Insights creates dynamic, automatic dashboards for container workloads running on Amazon ECS and Amazon EKS. You should enable Container Insights to have observability of CPU, memory, disk, network, and diagnostic information such as container restart failures. Container Insights generates dynamic dashboards that you can quickly filter at the cluster, container instance or node, service, task, pod, and individual container levels. Container Insights [is configured at the cluster and node or container instance level](#) depending on the AWS service.

Similar to Container Insights, CloudWatch Lambda Insights creates dynamic, automatic dashboards for your Lambda functions. This 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 and quickly resolve issues with your Lambda functions. Lambda is enabled at the function level and doesn't require any agents.

Container Insights and Lambda Insights also help you quickly switch to the application or performance logs, X-Ray traces, and a service map to visualize your container workloads. They both use the CloudWatch embedded metric format to capture CloudWatch metrics and performance logs.

You can create a shared CloudWatch dashboard for your workload that uses the metrics captured by Container Insights and Lambda Insights. You can do this by filtering and viewing the automatic dashboard through CloudWatch Container Insights and then choosing the **Add to Dashboard** option that allows you to add the metrics displayed to a standard CloudWatch dashboard. You can then remove or customize the metrics and add other metrics to correctly represent your workload.

CloudWatch integration with AWS services

AWS provides many services that include additional configuration options for logging and metrics. These services often enable you to configure CloudWatch Logs for log output and CloudWatch metrics for metrics output. The underlying infrastructure used to provide these services is managed by AWS and is inaccessible, but you can use the logging and metric options for your provisioned services to gain further insights and troubleshoot issues. For example, you can publish [VPC flow logs to CloudWatch](#), or you can also [configure Amazon Relational Database Service \(Amazon RDS\) instances to publish logs to CloudWatch](#).

Most AWS services log their API calls with [integration to AWS CloudTrail](#). CloudTrail also [supports integration with CloudWatch Logs](#) and this means that you can search and analyze activity in AWS services. You can also use Amazon CloudWatch Events or Amazon EventBridge to create and configure automation and notifications with CloudWatch Events event rules for specific actions performed in AWS services. Certain services [integrate directly](#) with CloudWatch Events and EventBridge. You can also [create events delivered through CloudTrail](#).

Amazon Managed Grafana for dashboarding and visualization

[Amazon Managed Grafana](#) can be used to observe and visualize your AWS workloads. Amazon Managed Grafana helps you visualize and analyze your operational data at scale. [Grafana](#) is an open-source analytics platform that helps you query, visualize, alert on, and understand your metrics wherever they are stored. Amazon Managed Grafana is particularly useful if your organization already uses Grafana for visualization of existing workloads and you want to extend coverage to AWS workloads. You can use Amazon Managed Grafana with CloudWatch by [adding it as a data source](#), which means that you can create visualizations using CloudWatch metrics. Amazon Managed Grafana supports AWS Organizations and you can centralize dashboards using CloudWatch metrics from multiple accounts and Regions.

The following table provides the advantages and considerations for using Amazon Managed Grafana instead of CloudWatch for dashboarding. A hybrid approach might be suitable based on the different requirements of your end users, workloads, and applications.

Create visualizations and dashboards that integrate with data sources supported by Amazon Managed Grafana and open-source Grafana

Amazon Managed Grafana helps you create visualizations and dashboards from many different data sources, including CloudWatch metrics. Amazon Managed Grafana includes a number of built-in data sources that span AWS services, open-source software, and COTS software. For more information about this, see [Built-in data sources](#) in the Amazon Managed Grafana documentation. You can also add support for more data sources by upgrading your workspace to [Grafana Enterprise](#). Grafana also supports [data source plugins](#) that allow you to communicate with different external systems. CloudWatch dashboards require a CloudWatch metric or CloudWatch Logs Insights query for data to be displayed on a CloudWatch dashboard.

Manage access to your dashboarding solution separately from your AWS account access

Amazon Managed Grafana requires the use of AWS IAM Identity Center (IAM Identity Center) and AWS Organizations for authentication and authorization. This enables you to authenticate users to Grafana by using identity federation that you might already use with IAM Identity Center or AWS Organizations. However, if you aren't using IAM Identity Center or AWS Organizations, then it is set up as a part of the Amazon Managed Grafana setup process. This might become an issue if your organization has limited the use of IAM Identity Center or AWS Organizations.

Ingest and access data across multiple accounts and Regions with AWS Organizations integration

Amazon Managed Grafana integrates with AWS Organizations to enable you to read data from AWS sources such as CloudWatch and Amazon OpenSearch Service across all your accounts. This makes it possible to create dashboards that display visualizations using data across your accounts. To automatically enable data access across AWS Organizations, you need to set up your Amazon Managed Grafana workspace in the AWS Organizations management account. This is not recommended based on [AWS Organizations best practices for the management account](#). In contrast, CloudWatch also [supports cross-account, cross-Region dashboards for CloudWatch metrics](#).

Use advanced visualization widgets and Grafana definitions available in the open-source community

Grafana provides a large collection of visualizations that you can use when creating your dashboards. There is also a large library of community-contributed dashboards that you can edit and reuse according to your requirements.

Use dashboards with new and existing Grafana deployments

If you already use Grafana, you can import and export dashboards from your Grafana deployments and customize them for use in Amazon Managed Grafana. Amazon Managed Grafana allows you to standardize on Grafana as your dashboarding solution.

Advanced setup and configuration for workspaces, permissions, and data sources

Amazon Managed Grafana enables you to create multiple Grafana workspaces that have their own set of configured data sources, users, and policies. This can help you meet more advanced use case requirements, as well as advanced security configurations. The advanced capabilities might require your teams to grow their experience with Grafana if they don't already have these skills.

Designing and implementing logging and monitoring with CloudWatch FAQ

This section provides answers to commonly raised questions about designing and implementing logging and monitoring solution with CloudWatch.

Where do I store my CloudWatch configuration files?

The CloudWatch agent for Amazon EC2 can apply multiple configuration files that are stored in the CloudWatch configuration directory. Ideally, you should store your CloudWatch configuration as a set of files because you can version control and use them again across multiple accounts and environments. For more information about this, see the [Managing CloudWatch configurations](#) section of this guide. Alternatively, you can store your configuration files in a repository on GitHub and automate the retrieval of the configuration files when a new EC2 instance is provisioned.

How can I create a ticket in my service management solution when an alarm is raised?

You integrate your service management system with an Amazon Simple Notification Service (Amazon SNS) topic and configure the CloudWatch alarm to notify the SNS topic when an alarm is raised. Your integrated system receives the SNS message and can create a ticket using your service management systems APIs or SDKs.

How do I use CloudWatch to capture log files in my containers?

Amazon ECS tasks and Amazon EKS pods can be configured to automatically send the STDOUT and STDERR output to CloudWatch. The recommended approach for logging containerized applications is to have containers send their output to STDOUT and STDERR. This is also covered in the [Twelve-Factor App manifesto](#).

However, if you want to send specific log files to CloudWatch then you can mount a volume in your Amazon EKS pod or Amazon ECS task definition to where your application will write its log files and use a sidecar container for Fluentd or Fluent Bit to send the logs to CloudWatch. You should consider symbolic linking a specific log file in your container to `/dev/stdout` and `/dev/`

stderr. For more information about this, see [View logs for a container or service](#) in the Docker documentation.

How do I monitor health issues for AWS services?

You can use the [AWS Health Dashboard](#) to monitor AWS health events. You can also refer to the [aws-health-tools](#) GitHub repository for sample automation solutions related to AWS health events.

How can I create a custom CloudWatch metric when no agent support exists?

You can use the embedded metric format to ingest metrics into CloudWatch. You can also use AWS SDK (for example, [put_metric_data](#)), AWS CLI (for example, [put-metric-data](#)), or AWS API (for example, [PutMetricData](#)) to create custom metrics. You should consider how any custom logic will be maintained long term. One approach would be to use Lambda with integrated embedded metric format support to create your metrics, along with a CloudWatch Events event [schedule rule](#) to establish the period for the metric.

How do I integrate my existing logging and monitoring tools with AWS?

You should refer to guidance provided by the software or service vendor for integrating with AWS. You might be able to use agent software, SDK, or an API provided to send logs and metrics to their solution. You might also be able to use an open-source solution, such as Fluentd or Fluent Bit, configured to the vendor's specifications. You can also use the AWS SDK and CloudWatch Logs subscription filters with Lambda and Kinesis Data Streams to create custom log processors and shippers. Finally, you should also consider how you will integrate the software if you are using multiple accounts and Regions.

Resources

Introduction

- [AWS Well-Architected](#)

Targeted business outcomes

- [logging-monitoring-apg-guide-examples](#)
- [Six advantages of cloud computing](#)

Planning your CloudWatch deployment

- [AWS Organizations terminology and concepts](#)
- [AWS Systems Manager Quick Setup](#)
- [Collecting metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent](#)
- [cloudwatch-config-s3-bucket.yaml](#)
- [Create the CloudWatch agent configuration file with the wizard](#)
- [Enterprise DevOps: Why you should run what you build](#)
- [Exporting log data to Amazon S3](#)
- [Fine-grained access control in Amazon OpenSearch Service](#)
- [Lambda quotas](#)
- [Manually create or edit the CloudWatch agent configuration file](#)
- [Real-time processing of log data with subscriptions](#)
- [Tools to build on AWS](#)

Configuring the CloudWatch agent for EC2 instances and on-premises servers

- [Amazon EC2 metric dimensions](#)

- [Burstable performance instances](#)
- [CloudWatch agent predefined metric sets](#)
- [Collect process metrics with the procstat plugin](#)
- [Configuring the CloudWatch agent for procstat](#)
- [Enable or turn off detailed monitoring for your instances](#)
- [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format](#)
- [Working with log groups and log streams](#)
- [List the available CloudWatch metrics for your instances](#)
- [PutLogEvents](#)
- [Retrieve custom metrics with collectd](#)
- [Retrieve custom metrics with StatsD](#)

CloudWatch agent installation approaches for Amazon EC2 and on-premises servers

- [Create an IAM service role for a hybrid environment](#)
- [Create a managed-instance activation for a hybrid environment](#)
- [Create IAM roles and users for use with the CloudWatch agent](#)
- [Download and configure the CloudWatch agent using the command line](#)
- [How can I configure on-premises servers that use Systems Manager agent and the unified CloudWatch agent to use only temporary credentials?](#)
- [Prerequisites for stack set operations](#)
- [Using spot instances](#)

Logging and monitoring on Amazon ECS

- [amazon-cloudwatch-logs-for-fluent-bit](#)
- [Amazon ECS CloudWatch metrics](#)
- [Amazon ECS Container Insights metrics](#)
- [Amazon ECS container agent](#)

- [Amazon ECS launch types](#)
- [Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS](#)
- [ecs_cluster_with_cloudwatch_linux.yaml](#)
- [ecs_cw_emf_example](#)
- [ecs_firelense_emf_example](#)
- [ecs-task-nginx-firelense.json](#)
- [Retrieving Amazon ECS optimized AMI metadata](#)
- [Using the awslogs log driver](#)
- [Using the client libraries to generate embedded metric format logs](#)

Logging and monitoring on Amazon EKS

- [Amazon EKS control plane logging](#)
- [amazon_eks_managed_node_group_launch_config.yaml](#)
- [Amazon EKS nodes](#)
- [amazon-eks-nodegroup.yaml](#)
- [Amazon EKS Service Level Agreement](#)
- [Container Insights Prometheus metrics monitoring](#)
- [Control plane metrics with Prometheus](#)
- [Deploy the Kubernetes Dashboard \(web UI\)](#)
- [Fargate logging](#)
- [Fluent Bit for Amazon EKS on Fargate](#)
- [How to capture application logs when using Amazon EKS on Fargate](#)
- [Installing the CloudWatch agent to collect Prometheus metrics](#)
- [Installing the Kubernetes Metrics Server](#)
- [kubernetes /dashboard](#)
- [Kubernetes Horizontal Pod Autoscaler](#)
- [Kubernetes Control Plane components](#)
- [Kubernetes pods](#)
- [Launch template support](#)

- [Managed node groups](#)
- [Managed node update behavior](#)
- [metrics-server](#)
- [Monitoring Amazon EKS on Fargate using Prometheus and Grafana](#)
- [prometheus_jmx](#)
- [prometheus / jmx_exporter](#)
- [Scraping additional Prometheus sources and importing those metrics](#)
- [Self-managed nodes](#)
- [Send logs to CloudWatch Logs](#)
- [Set up FluentD as a DaemonSet to send logs to CloudWatch Logs](#)
- [Set up Java/JMX sample workload on Amazon EKS and Kubernetes](#)
- [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics](#)
- [Vertical Pod Autoscaler](#)

Logging and metrics for AWS Lambda

- [Lambda invocation errors](#)
- [logging – Logging facility for Python](#)
- [Using the client libraries to generate embedded metric format logs](#)
- [Working with Lambda function metrics](#)

Searching and analyzing logs in CloudWatch

- [The Beats family](#)
- [Elastic Logstash](#)
- [Elastic Stack](#)
- [Streaming CloudWatch Logs data to Amazon OpenSearch Service](#)

Alarming options with CloudWatch

- [amazon-cloudwatch-auto-alarms](#)

- [AWS Service Management Connector for Jira Service Management](#)
- [AWS Service Management Connector for ServiceNow](#)

Monitoring application and service availability

- [Configuring DNS failover](#)

Tracing applications with AWS X-Ray

- [Amazon ECS task networking](#)
- [Configuring sampling rules in the X-Ray console](#)
- [Run Windows PowerShell commands or scripts](#)
- [Running the X-Ray daemon on Amazon EC2](#)
- [Sending trace data to X-Ray](#)
- [Service graph in X-Ray](#)

Dashboards and visualizations with CloudWatch

- [Amazon CloudWatch Metric Math simplifies near real-time monitoring of your Amazon EFS file systems](#)
- [Setting up CloudWatch Container Insights](#)
- [Using metric math](#)

CloudWatch integration with AWS services

- [AWS CloudTrail supported services and integrations](#)
- [CloudWatch Events event examples from supported services](#)
- [Events delivered via CloudTrail](#)
- [Monitoring CloudTrail log files with CloudWatch Logs](#)
- [Publishing database engine logs to CloudWatch Logs](#)
- [Publishing flow logs to CloudWatch Logs](#)

Amazon Managed Grafana for dashboarding and visualization

- [Best practices for the management account in AWS Organizations](#)
- [Built-in data sources for Amazon Managed Grafana](#)
- [Cross-account and cross-Region dashboards in CloudWatch](#)
- [Grafana plugins](#)

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an [RSS feed](#).

Change	Description	Date
Updated logging information	Updated the section about logging for AWS Lambda .	April 17, 2023
Updated configuration information	Updated and renamed the section about creating and storing CloudWatch configurations .	February 9, 2023
Updated metrics information	Updated the custom application metrics information in the Metrics for Amazon ECS section.	January 31, 2023
Removed preview notices	Amazon Managed Grafana is generally available.	May 25, 2022
Removed section	CloudWatch SDK Metrics is no longer supported.	January 7, 2022
Initial publication	—	April 30, 2021

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- **Refactor/re-architect** – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- **Replatform (lift and reshape)** – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- **Repurchase (drop and shop)** – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- **Rehost (lift and shift)** – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- **Relocate (hypervisor-level lift and shift)** – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on AWS.
- **Retain (revisit)** – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later

time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See [attribute-based access control](#).

abstracted services

See [managed services](#).

ACID

See [atomicity, consistency, isolation, durability](#).

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than [active-passive migration](#).

active-passive migration

A database migration method in which in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include SUM and MAX.

AI

See [artificial intelligence](#).

AIOps

See [artificial intelligence operations](#).

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to [the portfolio discovery and analysis process](#) and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see [What is Artificial Intelligence?](#)

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the [operations integration guide](#).

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see [ABAC for AWS](#) in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the [AWS CAF website](#) and the [AWS CAF whitepaper](#).

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

BCP

See [business continuity planning](#).

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see [Data in a behavior graph](#) in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also [endianness](#).

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as "Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see [About branches](#) (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the [Implement break-glass procedures](#) indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and [greenfield](#) strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the [Organized around business capabilities](#) section of the [Running containerized microservices on AWS](#) whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See [AWS Cloud Adoption Framework](#).

CCoE

See [Cloud Center of Excellence](#).

CDC

See [change data capture](#).

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use [AWS Fault Injection Service \(AWS FIS\)](#) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See [continuous integration and continuous delivery](#).

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the [CCoE posts](#) on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to [edge computing](#) technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see [Building your Cloud Operating Model](#).

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post [The Journey Toward Cloud-First & the Stages of Adoption](#) on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the [migration readiness guide](#).

CMDB

See [configuration management database](#).

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or AWS CodeCommit. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision

A field of AI used by machines to identify people, places, and things in images with accuracy at or above human levels. Often built with deep learning models, it automates extraction, analysis, classification, and understanding of useful information from a single image or a sequence of images.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in

an AWS account and Region, or across an organization, by using a YAML template. For more information, see [Conformance packs](#) in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see [Benefits of continuous delivery](#). CD can also stand for *continuous deployment*. For more information, see [Continuous Delivery vs. Continuous Deployment](#).

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see [Data classification](#).

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see [Building a data perimeter on AWS](#).

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See [database definition language](#).

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see [Services that work with AWS Organizations](#) in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See [environment](#).

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see [Detective controls](#) in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally

designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a [star schema](#), a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a [disaster](#). For more information, see [Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#) in the AWS Well-Architected Framework.

DML

See [database manipulation language](#).

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see [Modernizing legacy Microsoft ASP.NET \(ASMX\) web services incrementally by using containers and Amazon API Gateway](#).

DR

See [disaster recovery](#).

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to [detect drift in system resources](#), or you can use AWS Control Tower to [detect changes in your landing zone](#) that might affect compliance with governance requirements.

DVSM

See [development value stream mapping](#).

E

EDA

See [exploratory data analysis](#).

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with [cloud computing](#), edge computing can reduce communication latency and improve response time.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See [service endpoint](#).

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts

or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see [Create an endpoint service](#) in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see [Envelope encryption](#) in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the [program implementation guide](#).

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies,

and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a [star schema](#). It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see [AWS Fault Isolation Boundaries](#).

feature branch

See [branch](#).

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see [Machine learning model interpretability with :AWS](#).

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML

model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

FGAC

See [fine-grained access control](#).

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through [change data capture](#) to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

G

geo blocking

See [geographic restrictions](#).

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see [Restricting the geographic distribution of your content](#) in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the [trunk-based workflow](#) is the modern, preferred approach.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as [brownfield](#). If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See [high availability](#).

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. [AWS provides AWS SCT](#) that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

laC

See [infrastructure as code](#).

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See [industrial Internet of Things](#).

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently

more consistent, reliable, and predictable than [mutable infrastructure](#). For more information, see the [Deploy using immutable infrastructure](#) best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see [Building an industrial Internet of Things \(IIoT\) digital transformation strategy](#).

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see [What is IoT?](#)

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see [Machine learning model interpretability with AWS](#).

IoT

See [Internet of Things](#).

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the [operations integration guide](#).

ITIL

See [IT information library](#).

ITSM

See [IT service management](#).

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see [Setting up a secure and scalable multi-account AWS environment](#).

large migration

A migration of 300 or more servers.

LBAC

See [label-based access control](#).

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see [Apply least-privilege permissions](#) in the IAM documentation.

lift and shift

See [7 Rs](#).

little-endian system

A system that stores the least significant byte first. See also [endianness](#).

lower environments

See [environment](#).

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see [Machine Learning](#).

main branch

See [branch](#).

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

MAP

See [Migration Acceleration Program](#).

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see [Building mechanisms](#) in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see [Integrating microservices by using AWS serverless services](#).

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see [Implementing microservices on AWS](#).

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial

cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the [AWS migration strategy](#).

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the [discussion of migration factories](#) and the [Cloud Migration Factory guide](#) in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The [MPA tool](#) (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the [migration readiness guide](#). MRA is the first phase of the [AWS migration strategy](#).

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the [7 Rs](#) entry in this glossary and see [Mobilize your organization to accelerate large-scale migrations](#).

ML

See [machine learning](#).

MPA

See [Migration Portfolio Assessment](#).

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see [Strategy for modernizing applications in the AWS Cloud](#).

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see [Evaluating modernization readiness for applications in the AWS Cloud](#).

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see [Decomposing monoliths into microservices](#).

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of [immutable infrastructure](#) as a best practice.

O

OAC

See [origin access control](#).

OAI

See [origin access identity](#).

OCM

See [organizational change management](#).

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See [operations integration](#).

OLA

See [operational-level agreement](#).

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see [Operational Readiness Reviews \(ORR\)](#) in the AWS Well-Architected Framework.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the [operations integration guide](#).

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see [Creating a trail for an organization](#) in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the [OCM guide](#).

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic PUT and DELETE requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also [OAC](#), which provides more granular and enhanced access control.

ORR

See [operational readiness review](#).

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see [Permissions boundaries](#) in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See [personally identifiable information](#).

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

policy

An object that can define permissions (see [identity-based policy](#)), specify access conditions (see [resource-based policy](#)), or define the maximum permissions for all accounts in an organization in AWS Organizations (see [service control policy](#)).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see [Enabling data persistence in microservices](#).

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see [Evaluating migration readiness](#).

predicate

A query condition that returns true or false, commonly located in a WHERE clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see [Preventative controls](#) in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in [Roles terms and concepts](#) in the IAM documentation.

Privacy by Design

An approach in system engineering that takes privacy into account throughout the whole engineering process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see [Working with private hosted zones](#) in the Route 53 documentation.

proactive control

A [security control](#) designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the [Controls reference guide](#) in the AWS Control Tower documentation and see [Proactive controls](#) in *Implementing security controls on AWS*.

production environment

See [environment](#).

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See [responsible, accountable, consulted, informed \(RACI\)](#).

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See [responsible, accountable, consulted, informed \(RACI\)](#).

RCAC

See [row and column access control](#).

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See [7 Rs](#).

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See [7 Rs](#).

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see [Managing AWS Regions](#) in *AWS General Reference*.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See [7 Rs](#).

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See [7 Rs](#).

replatform

See [7 Rs](#).

repurchase

See [7 Rs](#).

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see [Responsive controls](#) in *Implementing security controls on AWS*.

retain

See [7 Rs](#).

retire

See [7 Rs](#).

rotation

The process of periodically updating a [secret](#) to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See [recovery point objective](#).

RTO

See [recovery time objective](#).

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see [About SAML 2.0-based federation](#) in the IAM documentation.

SCP

See [service control policy](#).

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see [Secret](#) in the Secrets Manager documentation.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: [preventative](#), [detective](#), [responsive](#), and [proactive](#).

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as [detective](#) or [responsive](#) security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see [Service control policies](#) in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see [AWS service endpoints](#) in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a [service-level indicator](#).

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see [Shared responsibility model](#).

SIEM

See [security information and event management system](#).

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See [service-level agreement](#).

SLI

See [service-level indicator](#).

SLO

See [service-level objective](#).

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid

innovation. For more information, see [Phased approach to modernizing applications in the AWS Cloud](#).

SPOF

See [single point of failure](#).

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a [data warehouse](#) or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was [introduced by Martin Fowler](#) as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see [Modernizing legacy Microsoft ASP.NET \(ASMX\) web services incrementally by using containers and Amazon API Gateway](#).

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use [Amazon CloudWatch Synthetics](#) to create these tests.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see [Tagging your AWS resources](#).

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See [environment](#).

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see [What is a transit gateway](#) in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see [Using AWS Organizations with other AWS services](#) in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the [Quantifying uncertainty in deep learning systems](#) guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See [environment](#).

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see [What is VPC peering](#) in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See [write once, read many](#).

WQF

See [AWS Workload Qualification Framework](#).

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered [immutable](#).

Z

zero-day exploit

An attack, typically malware, that takes advantage of a [zero-day vulnerability](#).

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.