User Guide

Amazon EC2 Auto Scaling

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon EC2 Auto Scaling: User Guide

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

What is Amazon EC2 Auto Scaling? ... 1
Features of Amazon EC2 Auto Scaling ... 1
Pricing for Amazon EC2 Auto Scaling ... 3
Get started ... 3
Work with Auto Scaling groups ... 4
Auto Scaling benefits ... 4
Example: Cover variable demand ... 5
Example: Web app architecture ... 7
Example: Distribute instances across Availability Zones ... 9
Instance lifecycle ... 12
Scale out ... 13
Instances in service ... 13
Scale in ... 14
Detach an instance ... 15
Attach an instance ... 15
Lifecycle hooks ... 15
Enter and exit standby ... 16
Amazon EC2 Auto Scaling quotas ... 16
Request throttling for the Amazon EC2 Auto Scaling API ... 18
EC2 termination rates ... 18
Other services ... 18
Set up ... 20
Prepare to use Amazon EC2 ... 20
Prepare to use the AWS CLI ... 20
Get started ... 21
Tutorial: Create your first Auto Scaling group ... 22
Prepare for the walkthrough ... 22
Step 1: Create a launch template ... 23
Step 2: Create a single-instance Auto Scaling group ... 24
Step 3: Verify your Auto Scaling group ... 25
Step 4: Terminate an instance in your Auto Scaling group ... 26
Step 5: Next steps ... 26
Step 6: Clean up ... 27
Tutorial: Set up a scaled and load-balanced application ... 28

Prerequisites ... 30
Step 1: Set up a launch template or launch configuration ... 30
Step 2: Create an Auto Scaling group ... 34
Step 3: Verify that your load balancer is attached ... 35
Step 4: Next steps ... 36
Step 5: Clean up ... 36
Related resources ... 38
Launch templates ... 39
Permissions to work with launch templates ... 40
API operations supported by launch templates ... 40
Create a launch template for an Auto Scaling group ... 40
Create your launch template (console) ... 41
Change the default network interface settings (console) ... 44
Modify the storage configuration (console) ... 46
Create a launch template from an existing instance (console) ... 48
Related resources ... 49
Limitations ... 49
Create a launch template using advanced settings ... 49
Required settings ... 50
Advanced settings ... 50
Request Spot Instances ... 55
Capacity Blocks for ML ... 56
Migrate your Auto Scaling groups to launch templates ... 61
Step 1: Find Auto Scaling groups that use launch configurations ... 62
Step 2: Copy a launch configuration to a launch template ... 64
Step 3: Update an Auto Scaling group to use a launch template ... 65
Step 4: Replace your instances ... 66
Additional information ... 67
Migrate CloudFormation stacks to launch templates ... 67
Find Auto Scaling groups that use a launch configuration ... 67
Update a stack to use a launch template ... 68
Understand update behavior of stack resources ... 72
Track the migration ... 73
Launch configuration mapping reference ... 73
AWS CLI examples for working with launch templates ... 75
Example usage ... 75

Create a basic launch template ... 76
Specify tags that tag instances at launch ... 77
Specify an IAM role to pass to instances ... 77
Assign public IP addresses ... 78
Specify a user data script that configures instances at launch ... 78
Specify a block device mapping ... 78
Specify Dedicated Hosts to bring software licenses from external vendors ... 79
Specify an existing network interface ... 79
Create multiple network interfaces ... 79
Manage your launch templates ... 80
Update an Auto Scaling group to use a launch template ... 83
Use Systems Manager parameters instead of AMI IDs ... 84
Create a launch template that specifies a parameter for the AMI ... 84
Verify a launch template gets the correct AMI ID ... 89
Related resources ... 90
Limitations ... 90
Launch configurations ... 92
Create a launch configuration ... 92
Create a launch configuration ... 93
Configure IMDS ... 96
Create a launch configuration using an EC2 instance ... 98
Change a launch configuration ... 102
Auto Scaling groups ... 104
Create Auto Scaling groups using launch templates ... 105
Create a group using a launch template ... 106
Create a group using the EC2 launch wizard ... 109
Use multiple instance types and purchase options ... 113
Create Auto Scaling groups using launch configurations ... 157
Create a group using a launch configuration ... 158
Create a group using an EC2 instance ... 161
Update an Auto Scaling group ... 167
Update Auto Scaling instances ... 168
Tag groups and instances ... 169
Tag naming and usage restrictions ... 170
EC2 instance tagging lifecycle ... 171
Tag your Auto Scaling groups ... 171

Delete tags ... 174
Tags for security ... 175
Control access to tags ... 176
Use tags to filter Auto Scaling groups ... 177
Instance maintenance policies ... 181
Overview ... 181
Set an instance maintenance policy on your group ... 188
Lifecycle hooks ... 193
Lifecycle hook availability ... 194
Considerations and limitations ... 194
Related resources ... 196
How lifecycle hooks work in Auto Scaling groups ... 197
Prepare to add a lifecycle hook ... 198
Retrieve the target lifecycle state ... 206
Add lifecycle hooks to your Auto Scaling group ... 208
Complete a lifecycle action in an Auto Scaling group ... 211
Tutorial: Use instance metadata to retrieve lifecycle state ... 213
Tutorial: Configure a lifecycle hook that invokes a Lambda function ... 222
Warm pools ... 231
Core concepts ... 231
Prerequisites ... 234
Update the instances in a warm pool ... 235
Related resources ... 235
Limitations ... 235
Use lifecycle hooks ... 236
Create a warm pool for an Auto Scaling group ... 240
View health check status ... 242
AWS CLI examples for working with warm pools ... 245
Detach-attach instances ... 248
Considerations for detaching instances ... 249
Considerations for attaching instances ... 249
Move an instance to a different group using detach and attach ... 250
Temporarily remove instances ... 255
How the standby state works ... 256
Considerations ... 256
Health status of an instance in a standby state ... 257

Temporarily remove an instance by setting it to standby ... 256
Delete your Auto Scaling infrastructure ... 262
Delete your Auto Scaling group ... 262
(Optional) Delete the launch configuration ... 263
(Optional) Delete the launch template ... 264
(Optional) Delete the load balancer and target groups ... 265
(Optional) Delete CloudWatch alarms ... 265
Replace your instances ... 267
Instance refresh ... 267
How an instance refresh works ... 268
Understand the default values ... 274
Start an instance refresh ... 277
Monitor an instance refresh ... 288
Cancel an instance refresh ... 291
Undo changes with a rollback ... 292
Use skip matching ... 297
Add checkpoints ... 307
Maximum instance lifetime ... 312
Considerations ... 313
Set the maximum instance lifetime ... 313
Limitations ... 315
Scale your group ... 316
Choose your scaling method ... 317
Set scaling limits ... 318
Set the default instance warmup ... 320
Scaling performance considerations ... 320
Choose the default instance warmup time ... 321
Enable the default instance warmup for a group ... 322
Verify the default instance warmup time for a group ... 324
Find scaling policies with a previously set instance warmup time ... 325
Clear the previously set instance warmup for a scaling policy ... 326
Manual scaling ... 326
Change the desired capacity of your Auto Scaling group ... 327
Terminate an instance in your Auto Scaling group (AWS CLI) ... 330
Scheduled scaling ... 332
How scheduled scaling works ... 333

Recurring schedules ... 333
Time zone ... 334
Considerations ... 334
Limitations ... 335
Create a scheduled action ... 335
View scheduled action details ... 337
Delete a scheduled action ... 338
Dynamic scaling ... 339
How dynamic scaling policies work ... 340
Multiple dynamic scaling policies ... 341
Target tracking scaling policies ... 343
Step and simple scaling policies ... 356
Scaling cooldowns ... 372
Scaling policy based on Amazon SQS ... 375
Verify a scaling activity ... 383
Disable a scaling policy ... 385
Delete a scaling policy for an Auto Scaling group ... 387
AWS CLI examples for scaling policies ... 390
Predictive scaling ... 393
How predictive scaling works ... 394
Create a predictive scaling policy ... 398
Evaluate your predictive scaling policies ... 405
Override the forecast ... 414
Use custom metrics ... 419
Control instance termination ... 430
Termination policy scenarios ... 431
Configure termination policies ... 435
Create a custom termination policy with Lambda ... 440
Use instance scale-in protection ... 447
Design for graceful instance termination ... 451
Suspend-resume processes ... 454
Types of processes ... 455
Considerations ... 456
Suspend processes ... 457
Resume processes ... 457
How suspended processes affect other processes ... 458

Monitor ... 462
Health checks ... 464
About health checks ... 465
Set the health check grace period ... 471
Monitor for impaired Amazon EBS volumes ... 474
Set up a custom health check ... 478
View the reason for health check failures ... 479
Troubleshoot unhealthy instances ... 481
Monitor with AWS Health Dashboard ... 484
Monitor CloudWatch metrics ... 485
View monitoring graphs in the Amazon EC2 Auto Scaling console ... 486
CloudWatch metrics for Amazon EC2 Auto Scaling ... 490
Configure monitoring for Auto Scaling instances ... 497
Log API calls using CloudTrail ... 499
Auto Scaling management events in CloudTrail ... 500
Auto Scaling event examples ... 501
Amazon SNS notification options ... 502
Amazon SNS and Amazon EC2 Auto Scaling ... 503
Work with other services ... 509
Capacity Rebalancing ... 509
Overview ... 510
Capacity Rebalancing behavior ... 511
Considerations ... 511
Enable Capacity Rebalancing ... 514
Capacity Reservations ... 520
Step 1: Create the Capacity Reservations ... 521
Step 2: Create a Capacity Reservation group ... 523
Step 3: Create a launch template ... 525
Step 4: Create an Auto Scaling group ... 527
Related resources ... 528
AWS CloudShell ... 529
AWS CloudFormation ... 529
Amazon EC2 Auto Scaling and AWS CloudFormation templates ... 530
Learn more about AWS CloudFormation ... 530
Compute Optimizer ... 531
Limitations ... 531

Findings ... 532
View recommendations ... 532
Considerations for evaluating the recommendations ... 533
Elastic Load Balancing ... 534
Elastic Load Balancing types ... 535
Prepare to attach a load balancer ... 536
Attach a load balancer ... 539
Configure a load balancer ... 543
Verify the attachment status ... 544
Add an Availability Zone ... 545
Remove an Availability Zone ... 547
Detach a load balancer ... 542
AWS CLI examples for working with Elastic Load Balancing ... 549
VPC Lattice ... 556
Prepare to attach a target group ... 559
Attach a VPC Lattice target group ... 562
Verify the attachment status ... 567
EventBridge ... 568
Amazon EC2 Auto Scaling event reference ... 568
Warm pool example events and patterns ... 579
EventBridge rules ... 585
Amazon VPC ... 590
Default VPC ... 590
Nondefault VPC ... 591
Considerations when choosing VPC subnets ... 591
IP addressing in a VPC ... 592
Network interfaces in a VPC ... 592
Instance placement tenancy ... 593
AWS Outposts ... 593
More resources for learning about VPCs ... 593
Security ... 595
Infrastructure security ... 595
Related resources ... 596
Resilience ... 596
Related resources ... 598
Data protection ... 598

Use AWS KMS keys to encrypt Amazon EBS volumes ... 599
Related resources ... 599
AWS KMS key policy for use with encrypted volumes ... 600
Identity and Access Management ... 606
Access control ... 606
How Amazon EC2 Auto Scaling works with IAM ... 607
API permissions ... 616
Managed policies ... 617
Service-linked roles ... 621
Identity-based policy examples ... 627
Cross-service confused deputy prevention ... 635
Control Amazon EC2 launch template usage in Auto Scaling groups ... 637
IAM role for applications that run on Amazon EC2 instances ... 646
Compliance validation ... 649
PCI DSS compliance ... 650
Use VPC endpoints for private connectivity ... 650
Create an interface VPC endpoint ... 651
Create a VPC endpoint policy ... 651
Working with AWS SDKs ... 653
Code examples ... 655
Basics ... 667
Hello Auto Scaling ... 669
Learn the basics ... 679
Actions ... 778
Scenarios ... 966
Build and manage a resilient service ... 966
Troubleshoot ... 1136
Retrieve an error message ... 1136
Turn off scaling activities ... 1138
Additional troubleshooting resources ... 1139
Instance launch failure ... 1140
The requested configuration is currently not supported. ... 1141
The security group <name of the security group> does not exist. Launching EC2 instance failed. ... 1141
The key pair <key pair associated with your EC2 instance> does not exist. Launching EC2 instance failed. ... 1142

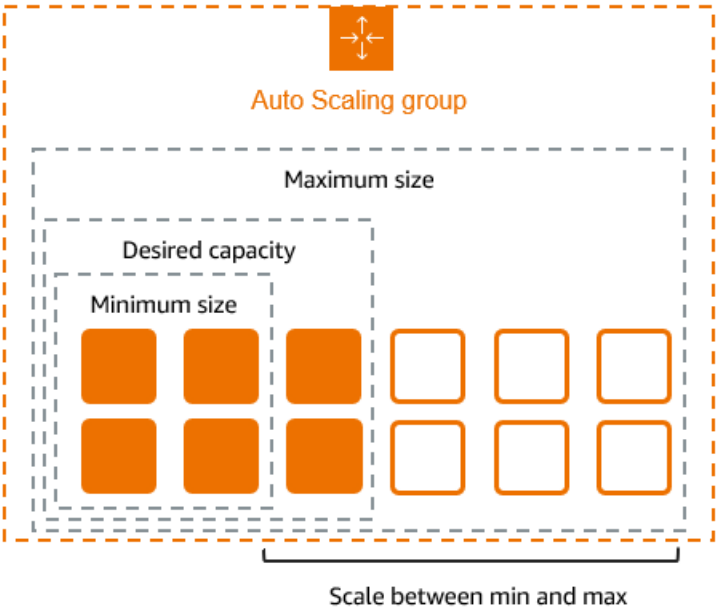
Your requested instance type (<instance type>) is not supported in your requested Availability Zone (<instance Availability Zone>)... 1142
Your Spot request price of 0.015 is lower than the minimum required Spot request fulfillment price of 0.0735... 1143
Invalid device name <device name> / Invalid device name upload. Launching EC2 instance failed. 1143
Value (<name associated with the instance storage device>) for parameter virtualName is invalid... Launching EC2 instance failed. 1144
EBS block device mappings not supported for instance-store AMIs. 1144
Placement groups may not be used with instances of type '<instance type>'. Launching EC2 instance failed. 1144
Client.InternalError: Client error on launch. 1145
We currently do not have sufficient <instance type> capacity in the Availability Zone you requested... Launching EC2 instance failed. 1146
The requested reservation does not have sufficient compatible and available capacity for this request. Launching EC2 instance failed. 1147
Your Capacity Block reservation <reservation id> is not active yet. Launching EC2 instance failed. 1147
There is no Spot capacity available that matches your request. Launching EC2 instance failed. 1148
<number of instances> instance(s) are already running. Launching EC2 instance failed. 1148
AMI issues 1148
The AMI ID <ID of your AMI> does not exist. Launching EC2 instance failed. 1149
AMI <AMI ID> is pending, and cannot be run. Launching EC2 instance failed. 1149
Invalid device name <device name>. Launching EC2 instance failed. 1150
The architecture 'arm64 ' of the specified instance type does not match the architecture 'x86_64' of the specified AMI...Launching EC2 instance failed. 1150
AMI '<AMI ID>' is disabled, and cannot be run. Launching EC2 instance failed. 1151
Load balancer issues 1152
One or more target groups not found. Validating load balancer configuration failed. 1153
Cannot find Load Balancer <your load balancer>. Validating load balancer configuration failed. 1153
There is no ACTIVE Load Balancer named <load balancer name>. Updating load balancer configuration failed. 1154
EC2 instance <instance ID> is not in VPC. Updating load balancer configuration failed. 1154
Launch template issues 1154

You must use a valid fully-formed launch template (invalid value) 1154
You are not authorized to use launch template (insufficient permissions) 1155
Related information 1157
Document history 1160

What is Amazon EC2 Auto Scaling?

Amazon EC2 Auto Scaling helps you ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of EC2 instances, called *Auto Scaling groups*. You can specify the minimum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes below this size. You can specify the maximum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes above this size. If you specify the desired capacity, either when you create the group or at any time thereafter, Amazon EC2 Auto Scaling ensures that your group has this many instances. If you specify scaling policies, then Amazon EC2 Auto Scaling can launch or terminate instances as demand on your application increases or decreases.

For example, the following Auto Scaling group has a minimum size of four instances, a desired capacity of six instances, and a maximum size of twelve instances. The scaling policies that you define adjust the number of instances, within your minimum and maximum number of instances, based on the criteria that you specify.



Features of Amazon EC2 Auto Scaling

With Amazon EC2 Auto Scaling, your EC2 instances are organized into Auto Scaling groups so that they can be treated as a logical unit for the purposes of scaling and management. Auto Scaling groups use launch templates (or launch configurations) as configuration templates for their EC2 instances.

The following are key features of Amazon EC2 Auto Scaling:

Monitoring the health of running instances

Amazon EC2 Auto Scaling automatically monitors the health and availability of your instances using EC2 health checks and replaces terminated or impaired instances to maintain your desired capacity.

Custom health checks

In addition to the built-in health checks, you can define custom health checks that are specific to your application to verify that it's responding as expected. If an instance fails your custom health check, it's automatically replaced to maintain your desired capacity.

Balancing capacity across Availability Zones

You can specify multiple Availability Zones for your Auto Scaling group, and Amazon EC2 Auto Scaling balances your instances evenly across the Availability Zones as the group scales. This provides high availability and resiliency by protecting your applications from failures in a single location.

Multiple instance types and purchase options

Within a single Auto Scaling group, you can launch multiple instance types and purchase options (Spot and On-Demand Instances), allowing you to optimize costs through Spot Instance usage. You can also take advantage of Reserved Instance and Savings Plan discounts by using them in conjunction with On-Demand Instances in the group.

Automated replacement of Spot Instances

If your group includes Spot Instances, Amazon EC2 Auto Scaling can automatically request replacement Spot capacity if your Spot Instances are interrupted. Through Capacity Rebalancing, Amazon EC2 Auto Scaling can also monitor and proactively replace your Spot Instances that are at an elevated risk of interruption.

Load balancing

You can use Elastic Load Balancing load balancing and health checks to ensure an even distribution of application traffic to your healthy instances. Whenever instances are launched or terminated, Amazon EC2 Auto Scaling automatically registers and deregisters the instances from the load balancer.

Scalability

Amazon EC2 Auto Scaling also provides several ways for you to scale your Auto Scaling groups. Using auto scaling allows you to maintain application availability and reduce costs by adding capacity to handle peak loads and removing capacity when demand is lower. You can also manually adjust the size of your Auto Scaling group as needed.

Instance refresh

The instance refresh feature provides a mechanism to update instances in a rolling fashion when you update your AMI or launch template. You can also use a phased approach, known as a canary deployment, to test a new AMI or launch template on a small set of instances before rolling it out to the whole group.

Lifecycle hooks

Lifecycle hooks are useful for defining custom actions that are invoked as new instances launch or before instances are terminated. This feature is particularly useful for building event-driven architectures, but it also helps you manage instances through their lifecycle.

Support for stateful workloads

Lifecycle hooks also offer a mechanism for persisting state on shut down. To ensure continuity for stateful applications, you can also use scale-in protection or custom termination policies to prevent instances with long-running processes from terminating early.

For more information about the benefits of Amazon EC2 Auto Scaling, see Auto Scaling benefits for application architecture.

Pricing for Amazon EC2 Auto Scaling

There are no additional fees with Amazon EC2 Auto Scaling, so it's easy to try it out and see how it can benefit your AWS architecture. You only pay for the AWS resources (for example, EC2 instances, EBS volumes, and CloudWatch alarms) that you use.

Get started

To begin, complete the Create your first Auto Scaling group tutorial to create an Auto Scaling group and see how it responds when an instance in that group terminates.

Work with Auto Scaling groups

You can create, access, and manage your Auto Scaling groups using any of the following interfaces:

- **AWS Management Console** – Provides a web interface that you can use to access your Auto Scaling groups. If you've signed up for an AWS account, you can access your Auto Scaling groups by signing into the AWS Management Console, using the search box on the navigation bar to search for **Auto Scaling groups**, and then choosing **Auto Scaling groups**.
- **AWS Command Line Interface (AWS CLI)** – Provides commands for a broad set of AWS services, and is supported on Windows, macOS, and Linux. To get started, see Prepare to use the AWS CLI. For more information, see autoscaling in the *AWS CLI Command Reference*.
- **AWS Tools for Windows PowerShell** – Provides commands for a broad set of AWS products for those who script in the PowerShell environment. To get started, see the AWS Tools for Windows PowerShell User Guide. For more information, see the AWS Tools for PowerShell Cmdlet Reference.
- **AWS SDKs** – Provides language-specific API operations and takes care of many of the connection details, such as calculating signatures, handling request retries, and handling errors. For more information, see AWS SDKs.
- **Query API** – Provides low-level API actions that you call using HTTPS requests. Using the Query API is the most direct way to access AWS services. However, it requires your application to handle low-level details such as generating the hash to sign the request, and handling errors. For more information, see the Amazon EC2 Auto Scaling API Reference.
- **AWS CloudFormation** – Supports creating Auto Scaling groups using CloudFormation templates. For more information, see Create Auto Scaling groups with AWS CloudFormation.

To connect programmatically to an AWS service, you use an endpoint. For information about endpoints for calls to Amazon EC2 Auto Scaling, see Amazon EC2 Auto Scaling endpoints and quotas in the *AWS General Reference*.

Auto Scaling benefits for application architecture

Adding Amazon EC2 Auto Scaling to your application architecture is one way to maximize the benefits of the AWS Cloud. When you use Amazon EC2 Auto Scaling, your applications gain the following benefits:

- Better fault tolerance. Amazon EC2 Auto Scaling can detect when an instance is unhealthy, terminate it, and launch an instance to replace it. You can also configure Amazon EC2 Auto Scaling to use multiple Availability Zones. If one Availability Zone becomes unavailable, Amazon EC2 Auto Scaling can launch instances in another one to compensate.
- Better availability. Amazon EC2 Auto Scaling helps ensure that your application always has the right amount of capacity to handle the current traffic demand.
- Better cost management. Amazon EC2 Auto Scaling can dynamically increase and decrease capacity as needed. Because you pay for the EC2 instances you use, you save money by launching instances when they are needed and terminating them when they aren't.

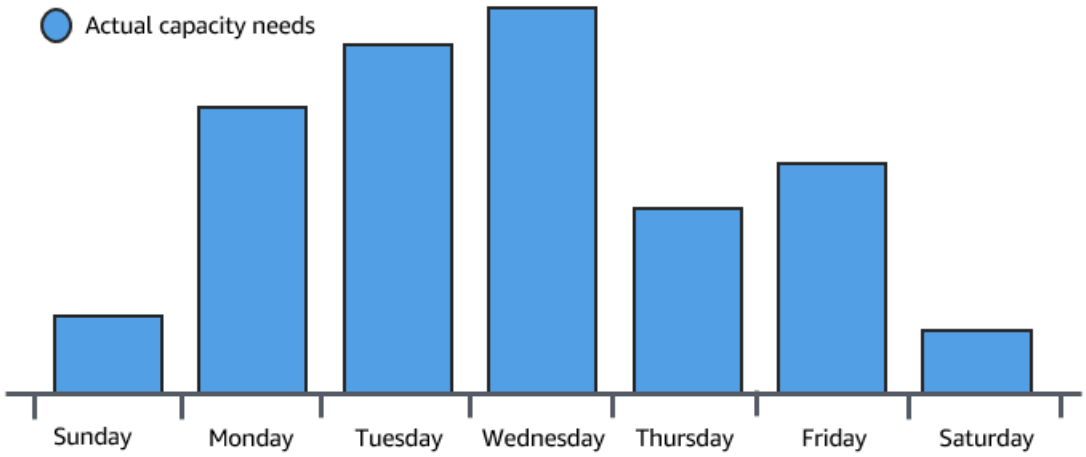
Contents

- Example: Cover variable demand
- Example: Web app architecture
- Example: Distribute instances across Availability Zones
  - Instance distribution
  - Rebalancing activities

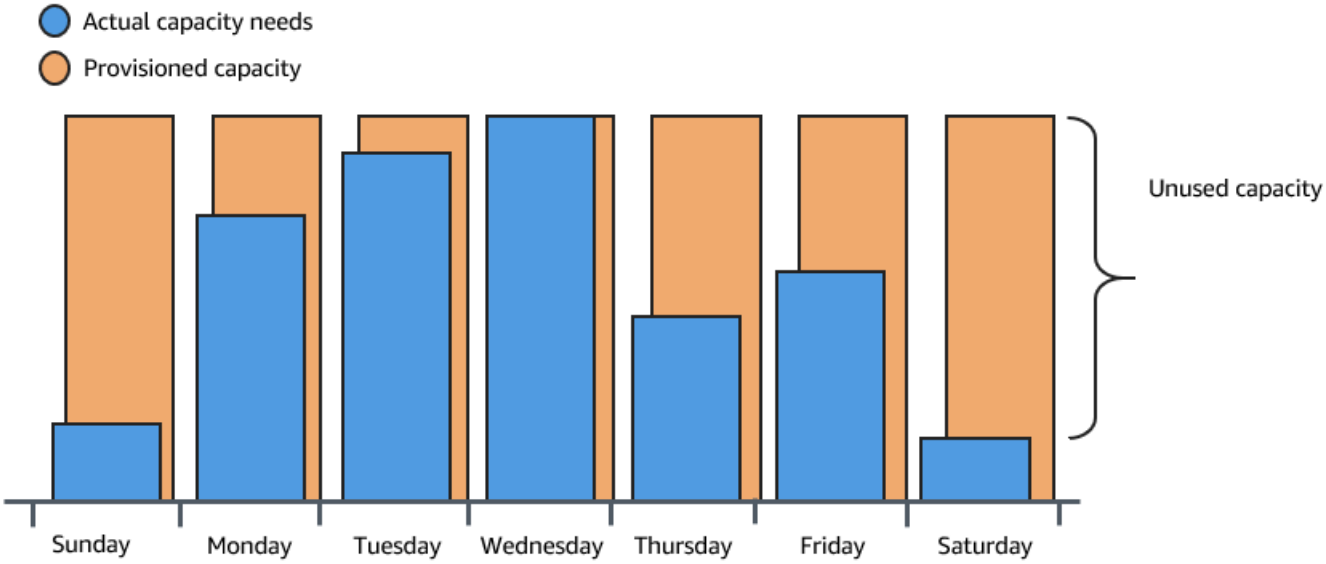
Example: Cover variable demand

To demonstrate some of the benefits of Amazon EC2 Auto Scaling, consider a basic web application running on AWS. This application allows employees to search for conference rooms that they might want to use for meetings. During the beginning and end of the week, usage of this application is minimal. During the middle of the week, more employees are scheduling meetings, so the demand on the application increases significantly.

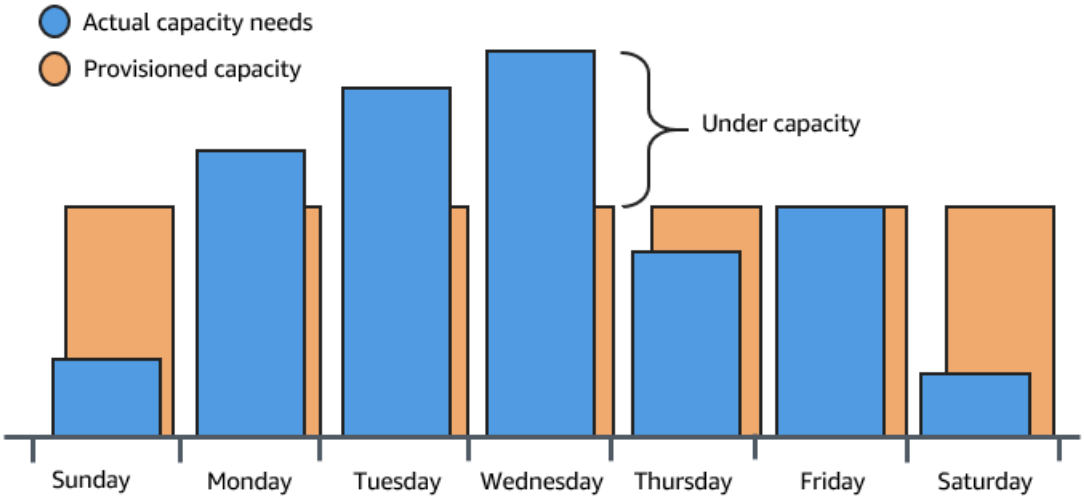
The following graph shows how much of the application's capacity is used over the course of a week.



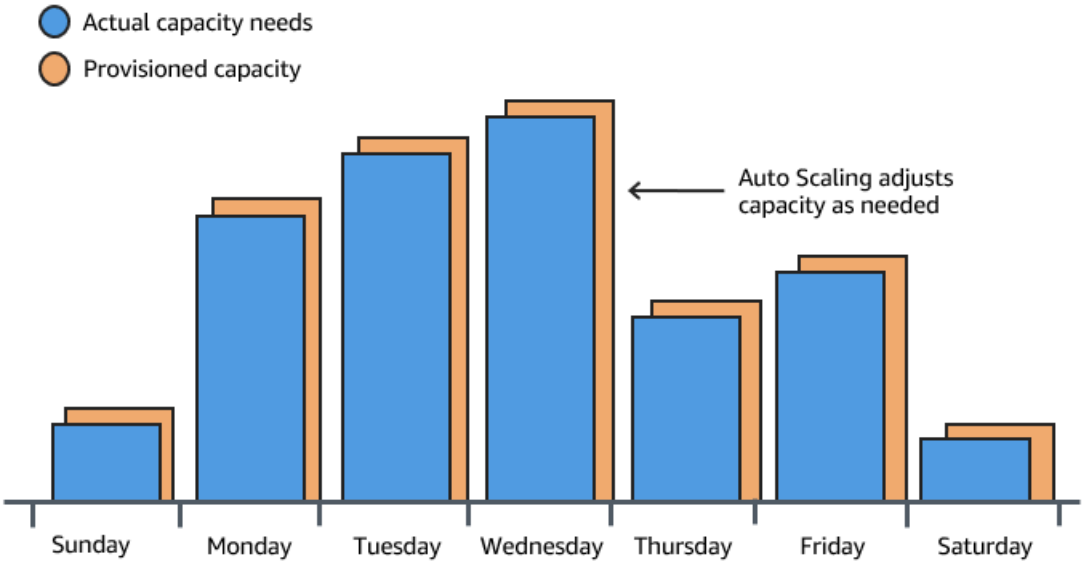
Traditionally, there are two ways to plan for these changes in capacity. The first option is to add enough servers so that the application always has enough capacity to meet demand. The downside of this option, however, is that there are days in which the application doesn't need this much capacity. The extra capacity remains unused and, in essence, raises the cost of keeping the application running.



The second option is to have enough capacity to handle the average demand on the application. This option is less expensive, because you aren't purchasing equipment that you'll only use occasionally. However, you risk creating a poor customer experience when the demand on the application exceeds its capacity.



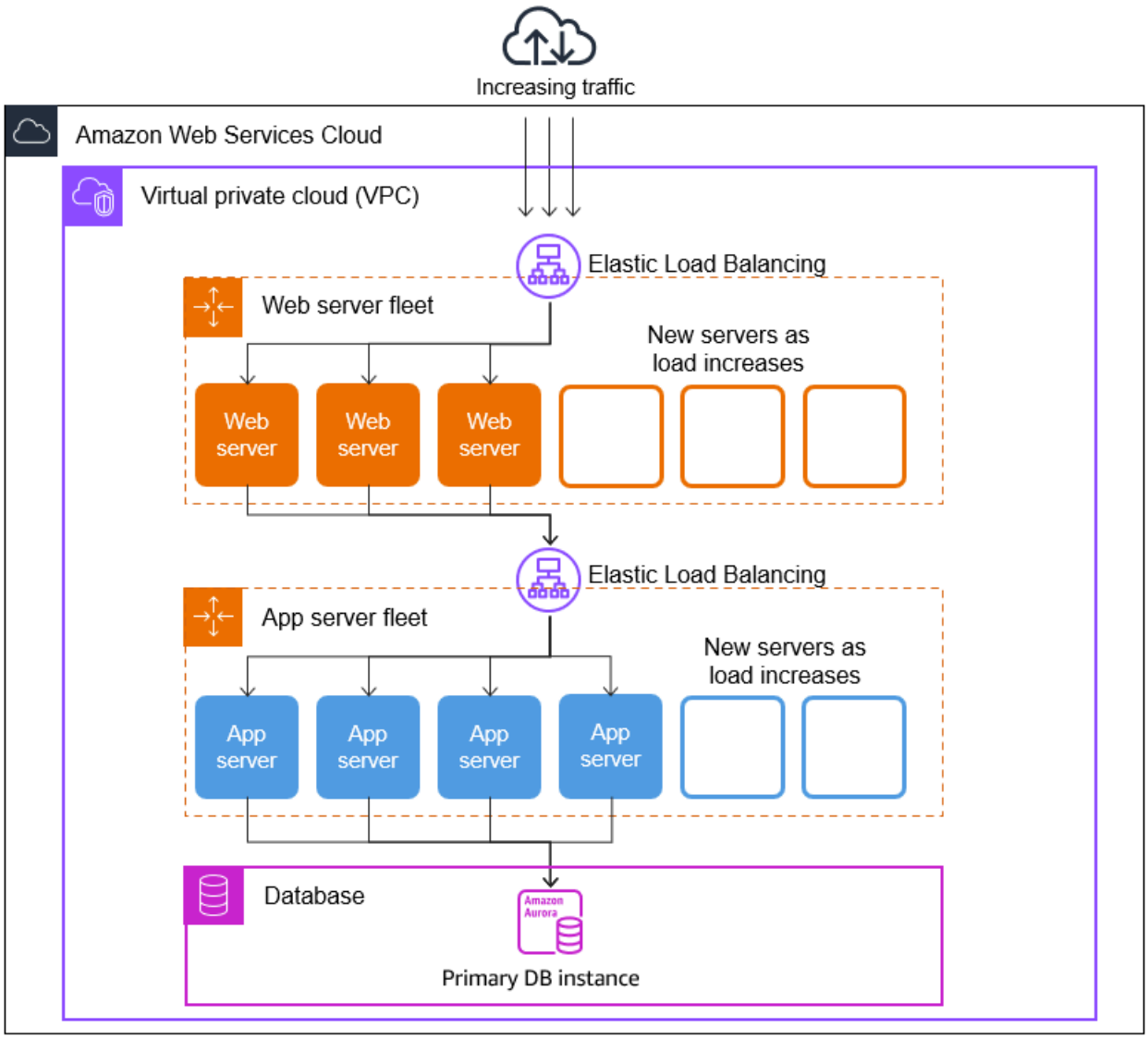
By adding Amazon EC2 Auto Scaling to this application, you have a third option available. You can add new instances to the application only when necessary, and terminate them when they're no longer needed. Because Amazon EC2 Auto Scaling uses EC2 instances, you only have to pay for the instances you use, when you use them. You now have a cost-effective architecture that provides the best customer experience while minimizing expenses.



Example: Web app architecture

In a common web app scenario, you run multiple copies of your app simultaneously to cover the volume of your customer traffic. These multiple copies of your application are hosted on identical EC2 instances (cloud servers), each handling customer requests.

Amazon EC2 Auto Scaling manages the launch and termination of these EC2 instances on your behalf. You define a set of criteria (such as an Amazon CloudWatch alarm) that determines when the Auto Scaling group launches or terminates EC2 instances. Adding Auto Scaling groups to your network architecture helps make your application more highly available and fault tolerant.



You can create as many Auto Scaling groups as you need. For example, you can create an Auto Scaling group for each tier.

To distribute traffic between the instances in your Auto Scaling groups, you can introduce a load balancer into your architecture. For more information, see Elastic Load Balancing.

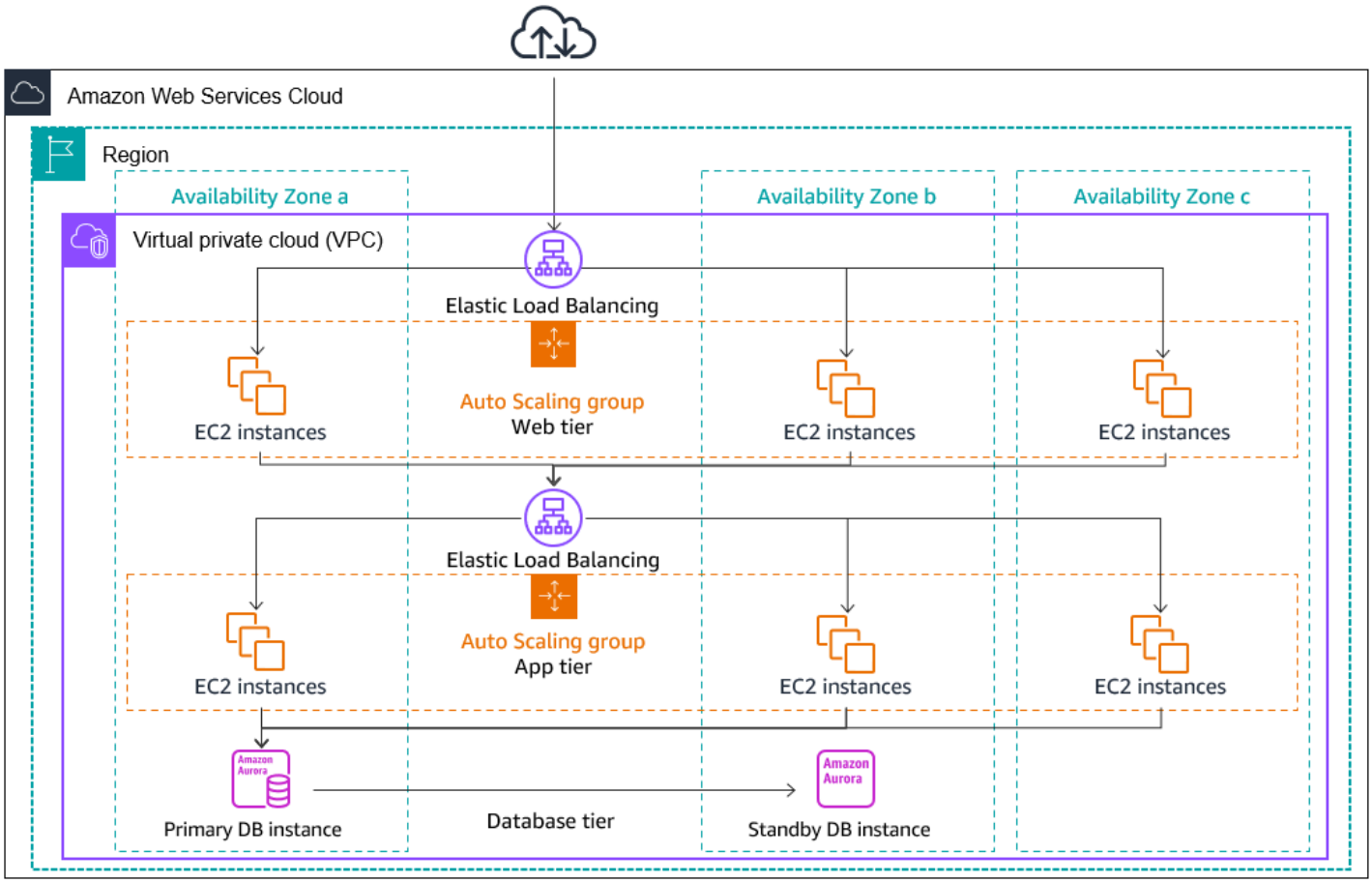
Example: Distribute instances across Availability Zones

Availability Zones are isolated locations in a given AWS Region. Each Region has multiple Availability Zones designed to provide high availability for the Region. Availability Zones are independent, and therefore you increase application availability when you design your application to use multiple zones. For more information, see Resilience in Amazon EC2 Auto Scaling.

An Availability Zone is identified by the AWS Region code followed by a letter identifier (for example, `us-east-1a`). If you create your VPC and subnets rather than using the default VPC, you can define one or more subnets in each Availability Zone. Each subnet must reside entirely within one Availability Zone and cannot span zones. For more information, see How Amazon VPC works in the *Amazon VPC User Guide*.

When you create an Auto Scaling group, you must choose the VPC and subnets where you will deploy the Auto Scaling group. Amazon EC2 Auto Scaling creates your instances in your chosen subnets. Each instance is thus associated with a specific Availability Zone chosen by Amazon EC2 Auto Scaling. When instances launch, Amazon EC2 Auto Scaling tries to evenly distribute them between the zones for high availability and reliability.

The following image shows an overview of multi-tier architecture deployed across three Availability Zones.



Instance distribution

Amazon EC2 Auto Scaling automatically tries to maintain equivalent numbers of instances in each enabled Availability Zone. Amazon EC2 Auto Scaling does this by attempting to launch new instances in the Availability Zone with the fewest instances. If there are multiple subnets chosen for the Availability Zone, Amazon EC2 Auto Scaling selects a subnet from the Availability Zone at random. If the attempt fails, however, Amazon EC2 Auto Scaling attempts to launch the instances in another Availability Zone until it succeeds.

In circumstances where an Availability Zone becomes unhealthy or unavailable, the distribution of instances might become unevenly distributed across the Availability Zones. When the Availability Zone recovers, Amazon EC2 Auto Scaling automatically rebalances the Auto Scaling group. It does this by launching instances in the enabled Availability Zones with the fewest instances and terminating instances elsewhere.

Rebalancing activities

Rebalancing activities fall into two categories: Availability Zone rebalancing and capacity rebalancing.

Availability Zone rebalancing

After certain actions occur, your Auto Scaling group can become unbalanced between Availability Zones. Amazon EC2 Auto Scaling compensates by rebalancing the Availability Zones. The following actions can lead to rebalancing activity:

- You change the Availability Zones associated with your Auto Scaling group.
- You explicitly terminate or detach instances or place instances in standby, and then the group becomes unbalanced.
- An Availability Zone that previously had insufficient capacity recovers and now has additional capacity.
- An Availability Zone that previously had a Spot price above your maximum price now has a Spot price below your maximum price.

When rebalancing, Amazon EC2 Auto Scaling launches new instances before terminating the earlier ones. This way, rebalancing does not compromise the performance or availability of your application.

Because Amazon EC2 Auto Scaling attempts to launch new instances before terminating the earlier ones, being at or near the specified maximum capacity could impede or completely halt rebalancing activities.

To avoid this problem, the system can temporarily exceed the specified maximum capacity of a group during a rebalancing activity. By default, it can do so by a margin of 10 percent or one instance, whichever is greater. The margin is extended only if the group is at or near maximum capacity and needs rebalancing. The extension lasts only as long as needed to rebalance the group (typically a few minutes).

Alternatively, you can establish thresholds for an Auto Scaling group by using an instance maintenance policy, and the group can only increase or decrease capacity within that threshold range. This way, you can control how quickly your group rebalances itself. For more information, see Instance maintenance policies.

Capacity Rebalancing

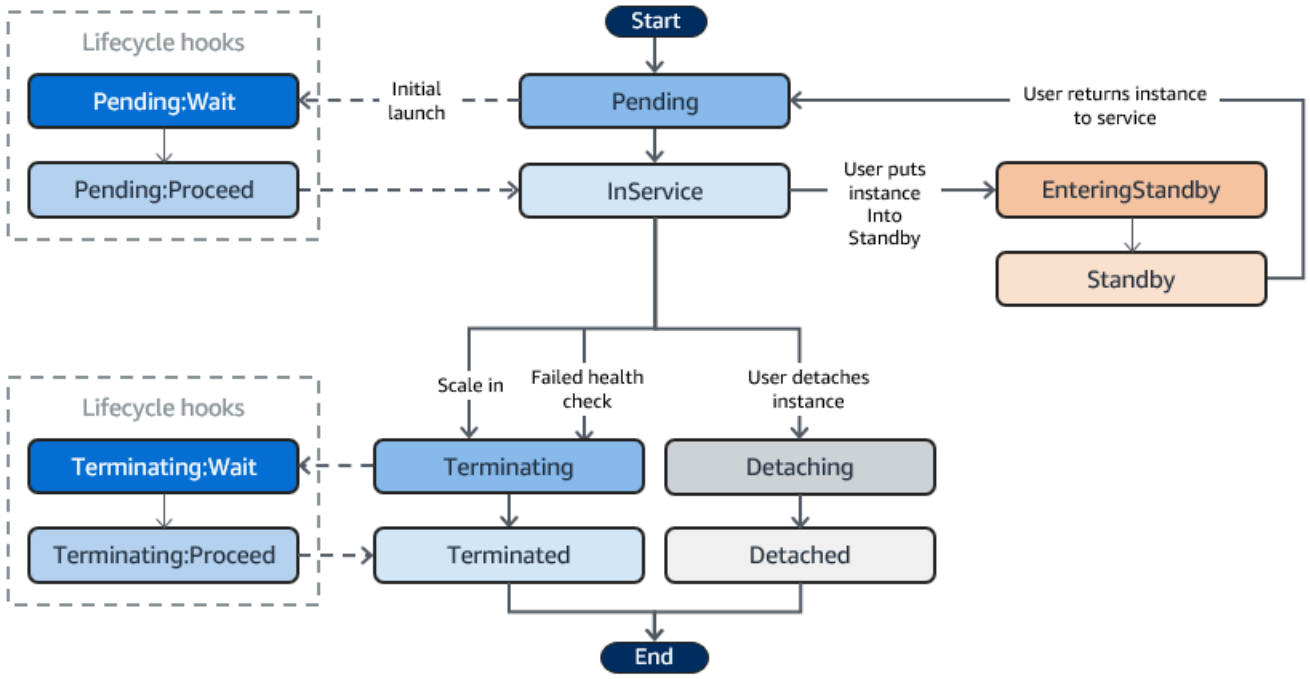
You can turn on Capacity Rebalancing for your Auto Scaling groups when using Spot Instances. This lets Amazon EC2 Auto Scaling attempt to launch a Spot Instance whenever Amazon EC2 reports that a Spot Instance is at an elevated risk of interruption. After launching a new instance, it then terminates an earlier instance. For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

Amazon EC2 Auto Scaling instance lifecycle

The EC2 instances in an Auto Scaling group have a path, or lifecycle, that differs from that of other EC2 instances. The lifecycle starts when the Auto Scaling group launches an instance and puts it into service. The lifecycle ends when you terminate the instance, or the Auto Scaling group takes the instance out of service and terminates it.

> **Note**
>
> You are billed for instances as soon as they are launched, including the time that they are not yet in service.

The following illustration shows the transitions between instance states in the Amazon EC2 Auto Scaling lifecycle.



Scale out

The following scale-out events direct the Auto Scaling group to launch EC2 instances and attach them to the group:

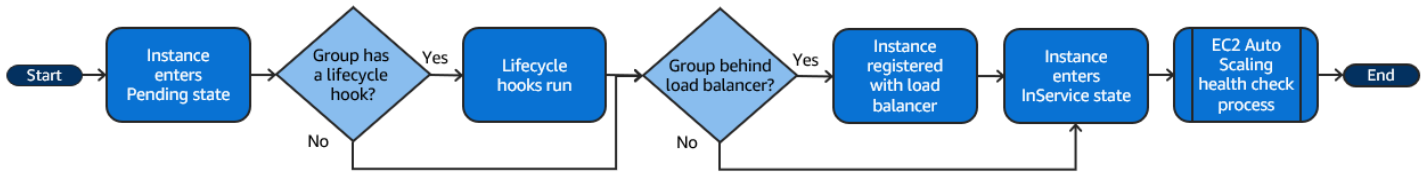
- You manually increase the size of the group. For more information, see Change the desired capacity of an existing Auto Scaling group.
- You create a scaling policy to automatically increase the size of the group based on a specified increase in demand. For more information, see Dynamic scaling for Amazon EC2 Auto Scaling.
- You set up scaling by schedule to increase the size of the group at a specific time. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling.

When a scale-out event occurs, the Auto Scaling group launches the required number of EC2 instances, using its assigned launch template. These instances start in the `Pending` state. If you add a lifecycle hook to your Auto Scaling group, you can perform a custom action here. For more information, see Lifecycle hooks.

When each instance is fully configured and passes the Amazon EC2 health checks, it is attached to the Auto Scaling group and it enters the `InService` state. The instance is counted against the desired capacity of the Auto Scaling group.

If your Auto Scaling group is configured to receive traffic from an Elastic Load Balancing load balancer, Amazon EC2 Auto Scaling automatically registers your instance with the load balancer before it marks the instance as `InService`.

The following summarizes the steps for registering an instance with a load balancer for a scale-out event.



Instances in service

Instances remain in the `InService` state until one of the following occurs:

- A scale-in event occurs, and Amazon EC2 Auto Scaling chooses to terminate this instance in order to reduce the size of the Auto Scaling group. For more information, see Control which Auto Scaling instances terminate during scale in.
- You put the instance into a `Standby` state. For more information, see Enter and exit standby.
- You detach the instance from the Auto Scaling group. For more information, see Detach or attach instances from your Auto Scaling group.
- The instance fails a required number of health checks, so it is removed from the Auto Scaling group, terminated, and replaced. For more information, see Health checks for instances in an Auto Scaling group.

Scale in

The following scale-in events direct the Auto Scaling group to detach EC2 instances from the group and terminate them:

- You manually decrease the size of the group. For more information, see Change the desired capacity of an existing Auto Scaling group.
- You create a scaling policy to automatically decrease the size of the group based on a specified decrease in demand. For more information, see Dynamic scaling for Amazon EC2 Auto Scaling.
- You set up scaling by schedule to decrease the size of the group at a specific time. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling.

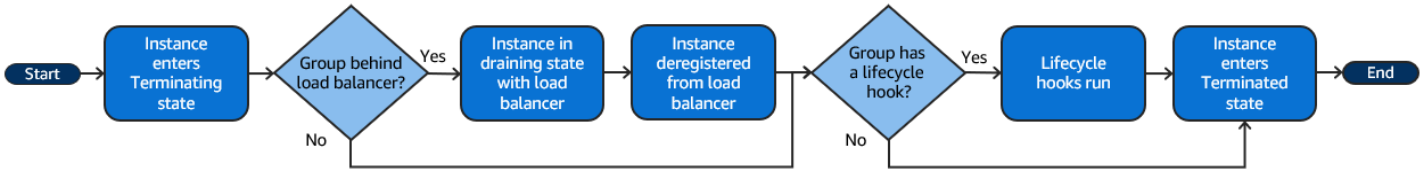
It is important that you create a corresponding scale-in event for each scale-out event that you create. This helps ensure that the resources assigned to your application match the demand for those resources as closely as possible.

When a scale-in event occurs, the Auto Scaling group terminates one or more instances. The Auto Scaling group uses its termination policy to determine which instances to terminate. Instances that are in the process of terminating from the Auto Scaling group enter the `Terminating` state, and can't be put back into service.

If your Auto Scaling group is configured to receive traffic from an Elastic Load Balancing load balancer, Amazon EC2 Auto Scaling automatically deregisters the terminating instance from the load balancer. Deregistering the instance ensures that all new requests are redirected to other instances in the load balancer's target group while existing connections to the instance are allowed to continue until the deregistration delay expires.

If you add a lifecycle hook to your Auto Scaling group, you can perform a custom action on the terminating instance. For more information, see Lifecycle hooks. Finally, the instance is completely terminated and enters the `Terminated` state.

The following summarizes the steps for deregistering an instance with a load balancer for a scale-in event.



Detach an instance

You can detach an instance from your Auto Scaling group. After the instance is detached, you can manage it separately from the Auto Scaling group or attach it to a different Auto Scaling group.

For more information, see Detach or attach instances from your Auto Scaling group.

Attach an instance

You can attach a running EC2 instance that meets certain criteria to your Auto Scaling group. After the instance is attached, it is managed as part of the Auto Scaling group.

For more information, see Detach or attach instances from your Auto Scaling group.

Lifecycle hooks

You can add a lifecycle hook to your Auto Scaling group so that you can perform custom actions when instances launch or terminate.

When Amazon EC2 Auto Scaling responds to a scale-out event, it launches one or more instances. These instances start in the `Pending` state. If you added an `autoscaling:EC2_INSTANCE_LAUNCHING` lifecycle hook to your Auto Scaling group, the instances move from the `Pending` state to the `Pending:Wait` state. After you complete the lifecycle action, the instances enter the `Pending:Proceed` state. When the instances are fully configured, they are attached to the Auto Scaling group and they enter the `InService` state.

When Amazon EC2 Auto Scaling responds to a scale-in event, it terminates one or more instances. These instances are detached from the Auto Scaling group and enter the `Terminating` state. If you added an `autoscaling:EC2_INSTANCE_TERMINATING` lifecycle hook to your Auto Scaling

group, the instances move from the `Terminating` state to the `Terminating:Wait` state. After you complete the lifecycle action, the instances enter the `Terminating:Proceed` state. When the instances are fully terminated, they enter the `Terminated` state.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks.

Enter and exit standby

You can put any instance that is in an `InService` state into a `Standby` state. This enables you to remove the instance from service, troubleshoot or make changes to it, and then put it back into service.

Instances in a `Standby` state continue to be managed by the Auto Scaling group. However, they are not an active part of your application until you put them back into service.

For more information, see Temporarily remove instances from your Auto Scaling group.

Quotas for Auto Scaling resources and groups

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

To view the quotas for Amazon EC2 Auto Scaling, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **Amazon EC2 Auto Scaling**.

To request a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*. If the quota is not yet available in Service Quotas, use the Auto Scaling Limits form. Quota increases are tied to the Region they were requested for.

Amazon EC2 Auto Scaling resources

Your AWS account has the following quotas related to the number of Auto Scaling groups and launch configurations that you can create.

Resource	Default quota
Auto Scaling groups per region	500
Launch configurations per region	200

Auto Scaling group configuration

Your AWS account has the following quotas related to the configuration of Auto Scaling groups. They cannot be changed.

Resource	Quota
Scaling policies per Auto Scaling group	50
Scheduled actions per Auto Scaling group	125
Step adjustments per step scaling policy	20
Lifecycle hooks per Auto Scaling group	50
SNS topics per Auto Scaling group	10
Classic Load Balancers per Auto Scaling group	50
Elastic Load Balancing target groups per Auto Scaling group	50
VPC Lattice target groups per Auto Scaling group	5

Auto Scaling group API operations

Amazon EC2 Auto Scaling provides API operations to make changes to your Auto Scaling groups in batches. The following are the API limits on the maximum number of items (maximum array members) that are allowed in a single operation. They cannot be changed.

Operation	Maximum array members
AttachInstances	20 instance IDs
AttachLoadBalancers	10 load balancers
AttachLoadBalancerTargetGroups	10 target groups
BatchDeleteScheduledAction	50 scheduled actions

Operation	Maximum array members
BatchPutScheduledUpdateGroupAction	50 scheduled actions
DetachInstances	20 instance IDs
DetachLoadBalancers	10 load balancers
DetachLoadBalancerTargetGroups	10 target groups
EnterStandby	20 instance IDs
ExitStandby	20 instance IDs
SetInstanceProtection	50 instance IDs

Request throttling for the Amazon EC2 Auto Scaling API

Amazon EC2 Auto Scaling API requests are throttled using a token bucket scheme to maintain service bandwidth. For more information, see API request rate in the *Amazon EC2 Auto Scaling API Reference*.

EC2 termination rates

Amazon EC2 Auto Scaling dynamically determines the number of EC2 instance termination operations it can perform at a time when your Auto Scaling group scales in. This means you might see variations in the number of instances terminated at a time across Auto Scaling groups. These variations are caused by external considerations, such as whether Amazon EC2 Auto Scaling must deregister instances with a load balancer.

Other services

Quotas for other services, such as Amazon EC2 and Amazon VPC, can impact your Auto Scaling groups. You can use Service Quotas to update the quotas for EC2 instances and other resources in your AWS account. In the Service Quotas console, you can view all your available service quotas and request increases for them. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

For quotas that are specific to launch templates, see Launch template restrictions in the *Amazon EC2 User Guide*.

Set up to use Amazon EC2 Auto Scaling

Before you start using Amazon EC2 Auto Scaling, complete the following tasks.

Tasks

- Prepare to use Amazon EC2
- Prepare to use the AWS CLI

Prepare to use Amazon EC2

If you haven't used Amazon EC2 before, complete the tasks described in the Amazon EC2 documentation. For more information, see Setting up with Amazon EC2 in the *Amazon EC2 User Guide* or Setting up with Amazon EC2 in the *Amazon EC2 User Guide*.

Prepare to use the AWS CLI

You can use the AWS command line tools to issue commands at your system's command line to perform Amazon EC2 Auto Scaling and other AWS tasks.

To use the AWS Command Line Interface (AWS CLI), download, install, and configure version 1 or 2 of the AWS CLI. The same Amazon EC2 Auto Scaling functionality is available in version 1 and 2. To install the AWS CLI version 1, see Installing, updating, and uninstalling the AWS CLI in the *AWS CLI Version 1 User Guide*. To install the AWS CLI version 2, see Installing or updating the latest version of the AWS CLI in the *AWS CLI Version 2 User Guide*.

AWS CloudShell lets you skip installing the AWS CLI in your development environment, and use it in the AWS Management Console instead. In addition to avoiding installation, you also don't need to configure credentials, and you don't need to specify a region. Your AWS Management Console session provides this context to the AWS CLI. You can use AWS CloudShell in supported AWS Regions. For more information, see Create Auto Scaling groups from the command line using AWS CloudShell.

For more information, see autoscaling in the *AWS CLI Command Reference*.

Get started with Amazon EC2 Auto Scaling

To get started with Amazon EC2 Auto Scaling, you can follow tutorials that introduce you to the service.

Topics

- Tutorial: Create your first Auto Scaling group
- Tutorial: Set up a scaled and load-balanced application

For additional tutorials that focus on specific tools for managing the lifecycle of instances in an Auto Scaling group, see the following topics:

- Tutorial: Configure a lifecycle hook that invokes a Lambda function. This tutorial shows you how to use Amazon EventBridge to create rules that invoke Lambda functions based on events that happen to the instances in your Auto Scaling group.
- Tutorial: Use data script and instance metadata to retrieve lifecycle state. This tutorial shows you how to use the Instance Metadata Service (IMDS) to invoke an action from within the instance itself.

Before you create an Auto Scaling group for use with your application, review your application thoroughly as it runs in the AWS Cloud. Consider the following:

- How many Availability Zones the Auto Scaling group should span.
- What existing resources can be used, such as security groups or Amazon Machine Images (AMIs).
- Whether you want to scale to increase or decrease capacity, or if you just want to ensure that a specific number of servers are always running. Keep in mind that Amazon EC2 Auto Scaling can do both simultaneously.
- What metrics have the most relevance to your application's performance.
- How long it takes to launch and provision a server.

The better you understand your application, the more effective you can make your Auto Scaling architecture.

Tutorial: Create your first Auto Scaling group

This tutorial provides a hands-on introduction to Amazon EC2 Auto Scaling through the AWS Management Console. You'll create a launch template that defines your EC2 instances and an Auto Scaling group with a single instance in it. After launching your Auto Scaling group, you'll terminate the instance and verify that the instance was removed from service and replaced. To maintain a constant number of instances, Amazon EC2 Auto Scaling detects and responds to Amazon EC2 health and reachability checks automatically.

When you sign up for AWS, you can get started with Amazon EC2 Auto Scaling for free using the AWS Free Tier. You can use the free tier to launch and use a `t2.micro` instance for free for 12 months (in Regions where `t2.micro` is unavailable, you can use a `t3.micro` instance under the free tier). If you launch an instance that is not within the free tier, you incur the standard Amazon EC2 usage fees for the instance. For more information, see Amazon EC2 pricing.

Tasks

- Prepare for the walkthrough
- Step 1: Create a launch template
- Step 2: Create a single-instance Auto Scaling group
- Step 3: Verify your Auto Scaling group
- Step 4: Terminate an instance in your Auto Scaling group
- Step 5: Next steps
- Step 6: Clean up

Prepare for the walkthrough

This walkthrough assumes that you are familiar with launching EC2 instances and that you have already created a key pair and a security group. For more information, see Setting up with Amazon EC2 in the *Amazon EC2 User Guide*.

To get started using Amazon EC2 Auto Scaling, you can use the *default* VPC for your AWS account. The default VPC includes a default public subnet in each Availability Zone and an internet gateway that is attached to your VPC. You can view your VPCs on the Your VPCs page of the Amazon Virtual Private Cloud (Amazon VPC) console.

Step 1: Create a launch template

In this step, you create a launch template that specifies the type of EC2 instance that Amazon EC2 Auto Scaling creates for you. Include information such as the ID of the Amazon Machine Image (AMI) to use, the instance type, the key pair, and security groups.

To create a launch template

1. Open the Amazon EC2 console and go to the Launch templates page.
2. On the top navigation bar, select an AWS Region. The launch template and Auto Scaling group that you create are tied to the Region that you specify.
3. Choose **Create launch template**.
4. For **Launch template name**, enter **`my-template-for-auto-scaling`**.
5. Under **Auto Scaling guidance**, select the check box.
6. For **Application and OS Images (Amazon Machine Image)**, choose a version of Amazon Linux 2 (HVM) from the **Quick Start** list. The AMI serves as a basic configuration template for your instances.
7. For **Instance type**, choose a hardware configuration that is compatible with the AMI that you specified.
8. (Optional) For **Key pair (login)**, choose an existing key pair. You use key pairs to connect to an Amazon EC2 instance with SSH. Connecting to an instance is not included as part of this tutorial. Therefore, you don't need to specify a key pair unless you intend to connect to your instance using SSH.
9. For **Network settings**, expand **Advanced network configuration** and do the following:
   a. Choose **Add network interface** to configure the primary network interface.
   b. For **Auto-assign public IP**, specify whether your instance receives a public IPv4 address. By default, Amazon EC2 assigns a public IPv4 address if the EC2 instance is launched into a default subnet or if the instance is launched into a subnet that's been configured to automatically assign a public IPv4 address. If you don't need to connect to your instance, choose **Disable**.
   c. For **Security group ID**, choose a security group in the same VPC that you plan to use as the VPC for your Auto Scaling group. If you don't specify a security group, your instance is automatically associated with the default security group for the VPC.

   d. For **Delete on termination**, choose **Yes** to delete the network interface when the instance is deleted.

10. Choose **Create launch template**.
11. On the confirmation page, choose **Create Auto Scaling group**.

Step 2: Create a single-instance Auto Scaling group

Use the following procedure to continue where you left off after creating a launch template.

To create an Auto Scaling group

1. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter `my-first-asg`.
2. Choose **Next**.

   The **Choose instance launch options** page appears, allowing you to choose the VPC network settings you want the Auto Scaling group to use and giving you options for launching On-Demand and Spot Instances.
3. In the **Network** section, keep **VPC** set to the default VPC for your chosen AWS Region, or select your own VPC. The default VPC is automatically configured to provide internet connectivity to your instance. This VPC includes a public subnet in each Availability Zone in the Region.
4. For **Availability Zones and subnets**, choose a subnet from each Availability Zone that you want to include. Use subnets in multiple Availability Zones for high availability. For more information, see Considerations when choosing VPC subnets.
5. In the **Instance type requirements** section, use the default setting to simplify this step. (Do not override the launch template.) For this tutorial, you will launch only one On-Demand Instance using the instance type specified in your launch template.
6. Keep the rest of the defaults for this tutorial and choose **Skip to review**.

   > **Note**
   >
   > The initial size of the group is determined by its desired capacity. The default value is 1 instance.

7. On the **Review** page, review the information for the group, and then choose **Create Auto Scaling group**.

Step 3: Verify your Auto Scaling group

Now that you have created an Auto Scaling group, you are ready to verify that the group has launched an EC2 instance.

> **Tip**
>
> In the following procedure, you look at the **Activity history** and **Instances** sections for the Auto Scaling group. In both, the named columns should already be displayed. To display hidden columns or change the number of rows shown, choose the gear icon on the top right corner of each section to open the preferences modal, update the settings as needed, and choose **Confirm**.

To verify that your Auto Scaling group has launched an EC2 instance

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to the Auto Scaling group that you just created.

   A split pane opens up in the bottom of the **Auto Scaling groups** page. The first tab available is the **Details** tab, showing information about the Auto Scaling group.
3. Choose the second tab, **Activity**. Under **Activity history**, you can view the progress of activities that are associated with the Auto Scaling group. The **Status** column shows the current status of your instance. While your instance is launching, the status column shows `Not yet in service`. The status changes to `Successful` after the instance is launched. You can also use the refresh button to see the current status of your instance.
4. On the **Instance management** tab, under **Instances**, you can view the status of the instance.
5. Verify that your instance launched successfully. It takes a short time for an instance to launch.

   - The **Lifecycle** column shows the state of your instance. Initially, your instance is in the `Pending` state. After an instance is ready to receive traffic, its state is `InService`.
   - The **Health status** column shows the result of the Amazon EC2 Auto Scaling health checks on your instance.

Step 4: Terminate an instance in your Auto Scaling group

Use these steps to learn more about how Amazon EC2 Auto Scaling works, specifically, how it launches new instances when necessary. The minimum size for the Auto Scaling group that you created in this tutorial is one instance. Therefore, if you terminate that running instance, Amazon EC2 Auto Scaling must launch a new instance to replace it.

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group.
3. On the **Instance management** tab, under **Instances**, select the ID of the instance.

   This takes you to the **Instances** page of the Amazon EC2 console, where you can terminate the instance.
4. Choose **Actions**, **Instance State**, **Terminate**. When prompted for confirmation, choose **Yes, Terminate**.
5. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**. Select your Auto Scaling group and choose the **Activity** tab.

   When you terminate an instance from the **Instances** page, it takes a minute or two after you terminate the instance before a new instance launches. In the activity history, when the scaling activity starts, you see an entry for the termination of the first instance and an entry for the launch of a new instance. Use the refresh button until you see the new entries.
6. On the **Instance management** tab, the **Instances** section shows the new instance only.
7. On the navigation pane, under **Instances**, choose **Instances**. This page shows both the terminated instance and the new running instance.

Step 5: Next steps

Go to the next step if you would like to delete the basic infrastructure that you just created. Otherwise, you can use this infrastructure as your base and try one or more of the following:

- Connect to your Linux instance using Session Manager or SSH. For more information, see Connect to your Linux instance using Session Manager and Connect to your Linux instance from Linux or macOS using SSH in the *Amazon EC2 User Guide*.
- Configure an Amazon SNS notification to notify you whenever your Auto Scaling group launches or terminates instances. For more information, see Amazon SNS notification options.

- Manually scale your Auto Scaling group to test the SNS notification. For more information, see Change the desired capacity of your Auto Scaling group.

You can also start familiarizing yourself with auto scaling concepts by reading about Target tracking scaling policies. If the load on your application changes, your Auto Scaling group can scale out (add instances) and scale in (run fewer instances) automatically by adjusting the desired capacity of the group between the minimum and maximum capacity limits. For more information about setting these limits, see Set scaling limits for your Auto Scaling group.

Step 6: Clean up

You can either delete your scaling infrastructure or delete just your Auto Scaling group and keep your launch template to use later.

If you launched an instance that is not within the AWS Free Tier, you should terminate your instance to prevent additional charges. When you terminate the instance, the data associated with it will also be deleted.

To delete your Auto Scaling group

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group (`my-first-asg`).
3. Choose **Delete**.
4. When prompted for confirmation, type **`delete`** to confirm deleting the specified Auto Scaling group and then choose **Delete**.

   A loading icon in the **Name** column indicates that the Auto Scaling group is being deleted. When the deletion has occurred, the **Desired**, **Min**, and **Max** columns show `0` instances for the Auto Scaling group. It takes a few minutes to terminate the instance and delete the group. Refresh the list to see the current state.

Skip the following procedure if you would like to keep your launch template.

To delete your launch template

1. Open the Launch templates page of the Amazon EC2 console.
2. Select your launch template (`my-template-for-auto-scaling`).

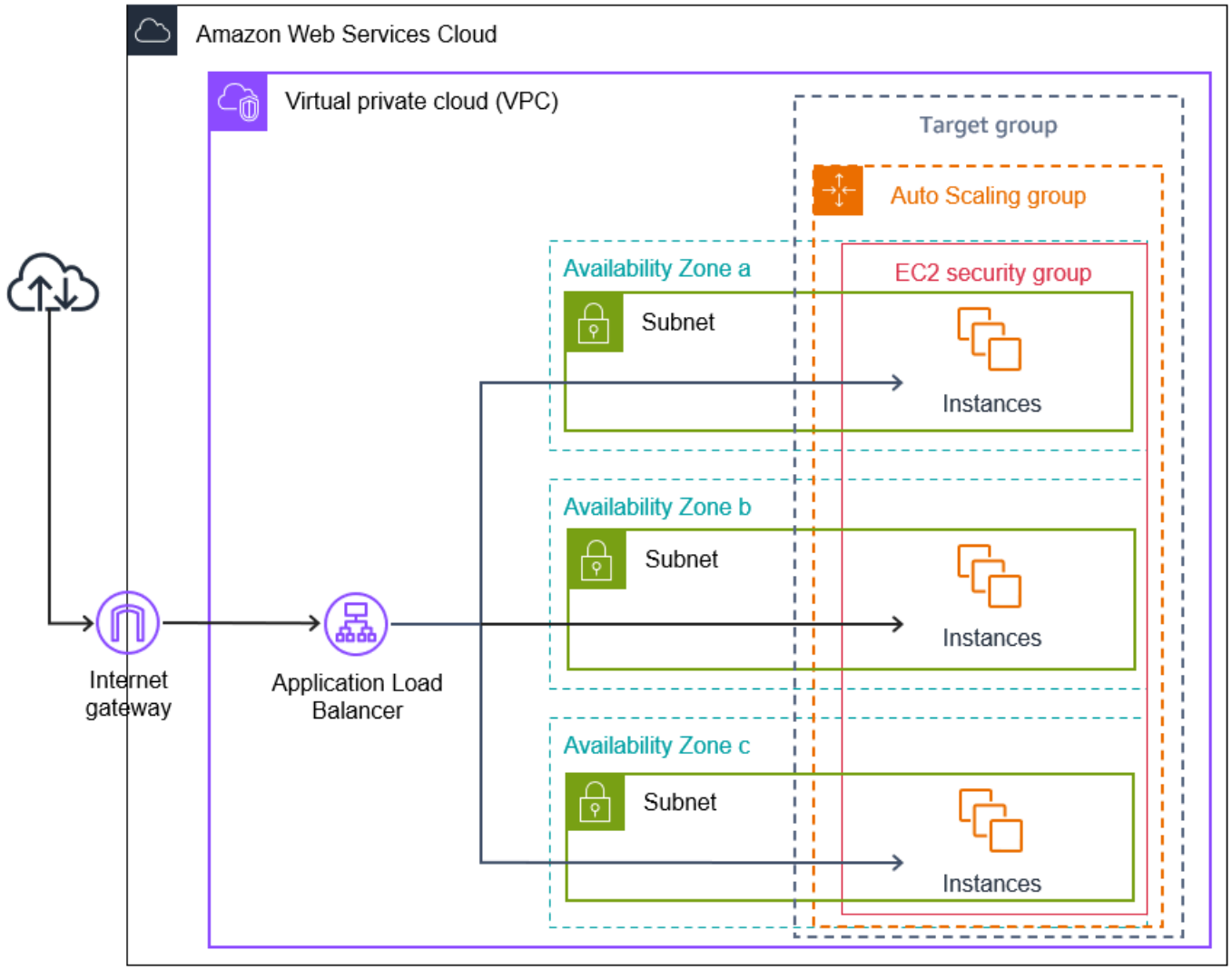
3. Choose **Actions**, **Delete template**.
4. When prompted for confirmation, type **`Delete`** to confirm deleting the specified launch template and then choose **Delete**.

Tutorial: Set up a scaled and load-balanced application

> **Important**
>
> Before you explore this tutorial, we recommend that you first review the following introductory tutorial: Create your first Auto Scaling group.

Registering your Auto Scaling group with an Elastic Load Balancing load balancer helps you set up a load-balanced application. Elastic Load Balancing works with Amazon EC2 Auto Scaling to distribute incoming traffic across your healthy Amazon EC2 instances. This increases the scalability and availability of your application. You can enable Elastic Load Balancing within multiple Availability Zones to increase the fault tolerance of your applications.

In this tutorial, we cover the basics steps for setting up a load-balanced application when the Auto Scaling group is created. When complete, your architecture should look similar to the following diagram:



Elastic Load Balancing supports different types of load balancers. We recommend that you use an Application Load Balancer for this tutorial.

For more information about introducing a load balancer into your architecture, see Use Elastic Load Balancing; to distribute incoming application traffic in your Auto Scaling group .

Tasks

- Prerequisites
- Step 1: Set up a launch template or launch configuration
- Step 2: Create an Auto Scaling group
- Step 3: Verify that your load balancer is attached
- Step 4: Next steps

- Step 5: Clean up
- Related resources

Prerequisites

- A load balancer and target group. Make sure to choose the same Availability Zones for the load balancer that you plan to use for your Auto Scaling group. For more information, see Getting started with Elastic Load Balancing in the *Elastic Load Balancing User Guide*.
- A security group for your launch template or launch configuration. The security group must allow access from the load balancer on both the listener port (usually port 80 for HTTP traffic) and the port that you want Elastic Load Balancing to use for health checks. For more information, see the applicable documentation:
  - Target security groups in the *User Guide for Application Load Balancers*
  - Target security groups in the *User Guide for Network Load Balancers*

  Optionally, if your instances will have public IP addresses, you can allow SSH traffic for connecting to the instances.
- (Optional) An IAM role that grants your application access to AWS.
- (Optional) An Amazon Machine Image (AMI) defined as the source template for your Amazon EC2 instances. To create one now, launch an instance. Specify the IAM role (if you created one) and any configuration scripts that you need as user data. Connect to the instance and customize it. For example, you can install software and applications, copy data, and attach additional EBS volumes. Test your applications on your instance to ensure that it is configured correctly. Save this updated configuration as a custom AMI. If you don't need the instance later, you can terminate it. Instances launched from this new custom AMI include the customizations that you made when you created the AMI.
- A virtual private cloud (VPC). This tutorial refers to the default VPC, but you can use your own. If using your own VPC, make sure that it has a subnet mapped to each Availability Zone of the Region you are working in. At minimum, you must have two public subnets available to create the load balancer. You must also have either two private subnets or two public subnets to create your Auto Scaling group and register it with the load balancer.

Step 1: Set up a launch template or launch configuration

Use either a launch template or a launch configuration for this tutorial.

Topics

- Select or create a launch template
- Select or create a launch configuration

Select or create a launch template

If you already have a launch template that you'd like to use, select it by using the following procedure.

To select an existing launch template

1. Open the Launch templates page of the Amazon EC2 console.
2. On the navigation bar at the top of the screen, choose the Region where the load balancer was created.
3. Select a launch template.
4. Choose **Actions**, **Create Auto Scaling group**.

Alternatively, to create a new launch template, use the following procedure.

To create a launch template

1. Open the Launch templates page of the Amazon EC2 console.
2. On the navigation bar at the top of the screen, choose the Region where the load balancer was created.
3. Choose **Create launch template**.
4. Enter a name and provide a description for the initial version of the launch template.
5. For **Application and OS Images (Amazon Machine Image)**, choose the ID of the AMI for your instances. You can search through all available AMIs, or select an AMI from the **Recents** or **Quick Start** list. If you don't see the AMI that you need, choose **Browse more AMIs** to browse the full AMI catalog.
6. For **Instance type**, select a hardware configuration for your instances that is compatible with the AMI that you specified.
7. (Optional) For **Key pair (login)**, choose the key pair to use when connecting to your instances.
8. For **Network settings**, expand **Advanced network configuration** and do the following:

a. Choose **Add network interface** to configure the primary network interface.

b. For **Auto-assign public IP**, specify whether your instances receive public IPv4 addresses. By default, Amazon EC2 assigns a public IPv4 address if the EC2 instance is launched into a default subnet or if the instance is launched into a subnet that's been configured to automatically assign a public IPv4 address. If you don't need to connect to your instances, you can choose **Disable** to prevent instances in your group from receiving traffic directly from the internet. In this case, they will receive traffic only from the load balancer.

c. For **Security group ID**, specify a security group for your instances from the same VPC as the load balancer.

d. For **Delete on termination**, choose **Yes**. This deletes the network interface when the Auto Scaling group scales in, and terminates the instance to which the network interface is attached.

9. (Optional) To securely distribute credentials to your instances, for **Advanced details**, **IAM instance profile**, enter the Amazon Resource Name (ARN) of your IAM role.
10. (Optional) To specify user data or a configuration script for your instances, paste it into **Advanced details**, **User data**.
11. Choose **Create launch template**.
12. On the confirmation page, choose **Create Auto Scaling group**.

Select or create a launch configuration

> **Note**
>
> We strongly discourage using launch configurations in new applications because it is a legacy feature with no planned investment. In addition, new accounts created on or after June 1, 2023 will not have the option to create new launch configurations through the console. For more information, see Auto Scaling launch configurations.

To select an existing launch configuration

1. Open the Launch configurations page of the Amazon EC2 console.
2. On the top navigation bar, choose the Region where the load balancer was created.
3. Select a launch configuration.

4. Choose **Actions**, **Create Auto Scaling group**.

Alternatively, to create a new launch configuration, use the following procedure.

To create a launch configuration

1. Open the Launch configurations page of the Amazon EC2 console. When prompted for confirmation, choose **View launch configurations** to confirm that you want to view the **Launch configurations** page.
2. On the top navigation bar, choose the Region where the load balancer was created.
3. Choose **Create launch configuration**, and enter a name for your launch configuration.
4. For **Amazon machine image (AMI)**, enter the ID of the AMI for your instances as search criteria.
5. For **Instance type**, select a hardware configuration for your instance.
6. Under **Additional configuration**, pay attention to the following fields:

   a. (Optional) To securely distribute credentials to your EC2 instance, for **IAM instance profile**, select your IAM role. For more information, see IAM role for applications that run on Amazon EC2 instances.

   b. (Optional) To specify user data or a configuration script for your instance, paste it into **Advanced details**, **User data**.

   c. (Optional) For **Advanced details**, **IP address type**, keep the default value. When you create your Auto Scaling group, you can assign a public IP address to instances in your Auto Scaling group by using subnets that have the public IP addressing attribute enabled, such as the default subnets in the default VPC. Alternatively, if you don't need to connect to your instances, you can choose **Do not assign a public IP address to any instances** to prevent instances in your group from receiving traffic directly from the internet. In this case, they will receive traffic only from the load balancer.
7. For **Security groups**, choose an existing security group from the same VPC as the load balancer. If you keep the **Create a new security group** option selected, a default SSH rule is configured for Amazon EC2 instances running Linux. A default RDP rule is configured for Amazon EC2 instances running Windows.
8. For **Key pair (login)**, choose an option under **Key pair options**.

   If you've already configured an Amazon EC2 instance key pair, you can choose it here.

If you don't already have an Amazon EC2 instance key pair, choose **Create a new key pair** and give it a recognizable name. Choose **Download key pair** to download the key pair to your computer.

> **Important**
>
> If you need to connect to your instances, do not choose **Proceed without a key pair**.

9. Select the acknowledgment check box, and then choose **Create launch configuration**.
10. Select the check box next to the name of your new launch configuration and choose **Actions**, **Create Auto Scaling group**.

Step 2: Create an Auto Scaling group

Use the following procedure to continue where you left off after creating or selecting your launch template or launch configuration.

To create an Auto Scaling group

1. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter a name for your Auto Scaling group.
2. [Launch template only] For **Launch template**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.
3. Choose **Next**.

   The **Choose instance launch options** page appears, allowing you to choose the VPC network settings you want the Auto Scaling group to use and giving you options for launching On-Demand and Spot Instances (if you chose a launch template).
4. In the **Network** section, for **VPC**, choose the VPC that you used for your load balancer. If you chose the default VPC, it is automatically configured to provide internet connectivity to your instances. This VPC includes a public subnet in each Availability Zone in the Region.
5. For **Availability Zones and subnets**, choose one or more subnets from each Availability Zone that you want to include, based on which Availability Zones the load balancer is in. For more information, see Considerations when choosing VPC subnets.

6. [Launch template only] In the **Instance type requirements** section, use the default setting to simplify this step. (Do not override the launch template.) For this tutorial, you will launch only On-Demand Instances using the instance type specified in your launch template.
7. Choose **Next** to go to the **Configure advanced options** page.
8. To attach the group to an existing load balancer, in the **Load balancing** section, choose **Attach to an existing load balancer**. You can choose **Choose from your load balancer target groups** or **Choose from Classic Load Balancers**. You can then choose the name of a target group for the Application Load Balancer or Network Load Balancer you created, or choose the name of a Classic Load Balancer.
9. (Optional) For **Health checks**, **Additional health check types**, select **Turn on Elastic Load Balancing health checks**.
10. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.
11. When you have finished configuring the Auto Scaling group, choose **Skip to review**.
12. On the **Review** page, review the details of your Auto Scaling group. You can choose **Edit** to make changes. When you are finished, choose **Create Auto Scaling group**.

After you have created the Auto Scaling group with the load balancer attached, the load balancer automatically registers new instances as they come online. You have only one instance at this point, so there isn't much to register. However, you can add additional instances by updating the desired capacity of the group. For step-by-step instructions, see Change the desired capacity of your Auto Scaling group.

Step 3: Verify that your load balancer is attached

To verify that your load balancer is attached

1. From the Auto Scaling groups page of the Amazon EC2 console, select the check box next to your Auto Scaling group.
2. On the **Details** tab, **Load balancing** shows any attached load balancer target groups or Classic Load Balancers.
3. On the **Activity** tab, in **Activity history**, you can verify that your instances launched successfully. The **Status** column shows whether your Auto Scaling group has successfully

launched instances. If your instances fail to launch, you can find troubleshooting ideas for common instance launch issues in Troubleshoot issues in Amazon EC2 Auto Scaling.

4. On the **Instance management** tab, under **Instances**, you can verify that your instances are ready to receive traffic. Initially, your instances are in the `Pending` state. After an instance is ready to receive traffic, its state is `InService`. The **Health status** column shows the result of the Amazon EC2 Auto Scaling health checks on your instances. Although an instance may be marked as healthy, the load balancer will only send traffic to instances that pass the load balancer health checks.
5. Verify that your instances are registered with the load balancer. Open the Target groups page of the Amazon EC2 console. Select your target group, and then choose the **Targets** tab. If the state of your instances is `initial`, it's probably because they are still in the process of being registered, or they are still undergoing health checks. When the state of your instances is `healthy`, they are ready for use.

Step 4: Next steps

Now that you have completed this tutorial, you can learn more:

- Amazon EC2 Auto Scaling determines whether an instance is healthy based on the status of the health checks that your Auto Scaling group uses. If you enable load balancer health checks and an instance fails the health checks, your Auto Scaling group considers the instance unhealthy and replaces it. For more information, see Health checks.
- You can expand your application to an additional Availability Zone in the same Region to increase fault tolerance if there is a service disruption. For more information, see ???.
- You can configure your Auto Scaling group to use a target tracking scaling policy. This automatically increases or decreases the number of instances as the demand on your instances changes. This allows the group to handle changes in the amount of traffic that your application receives. For more information, see Target tracking scaling policies.

Step 5: Clean up

After you're finished with the resources that you created for this tutorial, you should consider cleaning them up to avoid incurring unnecessary charges.

To delete your Auto Scaling group

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group.
3. Choose **Delete**.
4. When prompted for confirmation, type **`delete`** to confirm deleting the specified Auto Scaling group and then choose **Delete**.

   A loading icon in the **Name** column indicates that the Auto Scaling group is being deleted. When the deletion has occurred, the **Desired**, **Min**, and **Max** columns show `0` instances for the Auto Scaling group. It takes a few minutes to terminate the instance and delete the group. Refresh the list to see the current state.

Skip the following procedure if you would like to keep your launch template.

To delete your launch template

1. Open the Launch templates page of the Amazon EC2 console.
2. Select your launch template.
3. Choose **Actions**, **Delete template**.
4. When prompted for confirmation, type **`Delete`** to confirm deleting the specified launch template and then choose **Delete**.

Skip the following procedure if you would like to keep your launch configuration.

To delete your launch configuration

1. Open the Launch configurations page of the Amazon EC2 console.
2. Select your launch configuration.
3. Choose **Actions**, **Delete launch configuration**.
4. When prompted for confirmation, choose **Delete**.

Skip the following procedure if you want to keep the load balancer for future use.

To delete your load balancer

1. Open the Load balancers page of the Amazon EC2 console.
2. Choose the load balancer and choose **Actions**, **Delete**.
3. When prompted for confirmation, choose **Yes, Delete**.

To delete your target group

1. Open the Target groups page of the Amazon EC2 console.
2. Choose the target group and choose **Actions**, **Delete**.
3. When prompted for confirmation, choose **Yes, Delete**.

Related resources

With AWS CloudFormation, you can create and provision AWS infrastructure deployments predictably and repeatedly, by using template files to create and delete a collection of resources together as a single unit (a *stack*). For more information, see the AWS CloudFormation User Guide.

For a walkthrough that uses a stack template to provision an Auto Scaling group and Application Load Balancer, see Walkthrough: Create a scaled and load-balanced application in the *AWS CloudFormation User Guide*. Use the walkthrough and sample template as a starting point to create similar templates to meet your needs.

Auto Scaling launch templates

A launch template is similar to a launch configuration, in that it specifies instance configuration information. It includes the ID of the Amazon Machine Image (AMI), the instance type, a key pair, security groups, and other parameters used to launch EC2 instances. However, defining a launch template instead of a launch configuration allows you to have multiple versions of a launch template.

With versioning of launch templates, you can create a subset of the full set of parameters. Then, you can reuse it to create other versions of the same launch template. For example, you can create a launch template that defines a base configuration without an AMI or user data script. After you create your launch template, you can create a new version and add the AMI and user data that has the latest version of your application for testing. This results in two versions of the launch template. Storing a base configuration helps you to maintain the required general configuration parameters. You can create a new version of your launch template from the base configuration whenever you want. You can also delete the versions used for testing your application when you no longer need them.

We recommend that you use launch templates to ensure that you're accessing the latest features and improvements. Not all Amazon EC2 Auto Scaling features are available when you use launch configurations. For example, you cannot create an Auto Scaling group that launches both Spot and On-Demand Instances or that specifies multiple instance types. You must use a launch template to configure these features. For more information, see Auto Scaling groups with multiple instance types and purchase options.

With launch templates, you can also use newer features of Amazon EC2. This includes Systems Manager parameters (AMI ID), the current generation of EBS Provisioned IOPS volumes (io2), EBS volume tagging, T2 Unlimited instances, Capacity Reservations, Capacity Blocks, and Dedicated Hosts, to name a few.

When you create a launch template, all parameters are optional. However, if a launch template does not specify an AMI, you cannot add the AMI when you create your Auto Scaling group. If you specify an AMI but no instance type, you can add one or more instance types when you create your Auto Scaling group.

Contents

- Permissions to work with launch templates
- API operations supported by launch templates

- Create a launch template for an Auto Scaling group
- Create a launch template using advanced settings
- Migrate your Auto Scaling groups to launch templates
- Migrate AWS CloudFormation stacks to launch templates
- Examples for creating and managing launch templates with the AWS CLI
- Use AWS Systems Manager parameters instead of AMI IDs in launch templates

Permissions to work with launch templates

The procedures in this section assume that you already have the required permissions to create launch templates. For information about how an administrator grants you permissions, see Control access to launch templates with IAM permissions in the *Amazon EC2 User Guide*.

Note that if you do not have sufficient permissions to use and create resources specified in a launch template, you receive an error that you're not authorized to use the launch template when you try to specify it for an Auto Scaling group. For more information, see Troubleshoot Amazon EC2 Auto Scaling: Launch templates.

For examples of IAM policies that let you call the `CreateAutoScalingGroup`, `UpdateAutoScalingGroup`, and `RunInstances` API operations with a launch template, see Control Amazon EC2 launch template usage in Auto Scaling groups.

API operations supported by launch templates

For a list of API operations supported by launch templates, see Amazon EC2 actions in the *Amazon EC2 API Reference*.

Create a launch template for an Auto Scaling group

Before you can create an Auto Scaling group using a launch template, you must create a launch template that contains the configuration information to launch an instance, including the ID of the Amazon Machine Image (AMI).

To create new launch templates, use the following procedures.

Contents

- Create your launch template (console)

- Change the default network interface settings (console)
- Modify the storage configuration (console)
- Create a launch template from an existing instance (console)
- Related resources
- Limitations

> **Important**
>
> Launch template parameters are not fully validated when you create the launch template. If you specify incorrect values for parameters, or if you do not use supported parameter combinations, no instances can launch using this launch template. Be sure to specify the correct values for the parameters and use supported parameter combinations. For example, to launch instances with an Arm-based AWS Graviton or Graviton2 AMI, you must specify an Arm-compatible instance type. For more information, see Launch template restrictions in the *Amazon EC2 User Guide*.

Create your launch template (console)

The following steps describe how to configure a basic launch template:

- Specify the Amazon Machine Image (AMI) from which to launch the instances.
- Choose an instance type that is compatible with the AMI that you specify.
- Specify the key pair to use when connecting to instances, for example, using SSH.
- Add one or more security groups to allow network access to the instances.
- Specify whether to attach additional volumes to each instance.
- Add custom tags (key-value pairs) to the instances and volumes.

To create a launch template

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Instances**, choose **Launch Templates**.
3. Choose **Create launch template**. Enter a name and provide a description for the initial version of the launch template.

4. (Optional) Under **Auto Scaling guidance**, select the check box to have Amazon EC2 provide guidance to help create a template to use with Amazon EC2 Auto Scaling.
5. Under **Launch template contents**, fill out each required field and any optional fields as needed.

   a. **Application and OS Images (Amazon Machine Image)**: (Required) Choose the ID of the AMI for your instances. You can search through all available AMIs, or select an AMI from the **Recents** or **Quick Start** list. If you don't see the AMI that you need, choose **Browse more AMIs** to browse the full AMI catalog.

      To choose a custom AMI, you must first create your AMI from a customized instance. For more information, see Create an Amazon EBS-backed AMI in the *Amazon EC2 User Guide*.

   b. For **Instance type**, choose a single instance type that's compatible with the AMI that you specified.

      Alternatively, to use attribute-based instance type selection, choose **Advanced**, **Specify instance type attributes**, and then specify the following options:

      - **Number of vCPUs**: Enter the minimum and maximum number of vCPUs. To indicate no limits, enter a minimum of 0, and keep the maximum blank.
      - **Amount of memory (MiB)**: Enter the minimum and maximum amount of memory, in MiB. To indicate no limits, enter a minimum of 0, and keep the maximum blank.
      - Expand **Optional instance type attributes** and choose **Add attribute** to further limit the types of instances that can be used to fulfill your desired capacity. For information about each attribute, see InstanceRequirementsRequest in the *Amazon EC2 API Reference*.
      - **Resulting instance types**: You can view the instance types that match the specified compute requirements, such as vCPUs, memory, and storage.
      - To exclude instance types, choose **Add attribute**. From the **Attribute** list, choose **Excluded instance types**. From the **Attribute value** list, select the instance types to exclude.

   c. **Key pair (login)**: For **Key pair name**, choose an existing key pair, or choose **Create new key pair** to create a new one. For more information, see Amazon EC2 key pairs and Linux instances in the *Amazon EC2 User Guide*.

   d. **Network settings**: For **Firewall (security groups)**, use one or more security groups, or keep this blank and configure one or more security groups as part of the network

interface. For more information, see Amazon EC2 security groups for Linux instances in the *Amazon EC2 User Guide*.

If you don't specify any security groups in your launch template, Amazon EC2 uses the default security group for the VPC that your Auto Scaling group will launch instances into. By default, this security group doesn't allow inbound traffic from external networks. For more information, see Default security groups for your VPCs in the *Amazon VPC User Guide*.

e. Do one of the following:

- Change the default network interface settings. For example, you can enable or disable the public IPv4 addressing feature, which overrides the auto-assign public IPv4 addresses setting on the subnet. For more information, see Change the default network interface settings (console).
- Skip this step to keep the default network interface settings.

f. Do one of the following:

- Modify the storage configuration. For more information, see Modify the storage configuration (console).
- Skip this step to keep the default storage configuration.

g. For **Resource tags**, specify tags by providing key and value combinations. If you specify instance tags in your launch template and then you choose to propagate your Auto Scaling group's tags to its instances, all the tags are merged. If the same tag key is specified for a tag in your launch template and a tag in your Auto Scaling group, then the tag value from the group takes precedence.

6. (Optional) Configure advanced settings. For example, you can choose an IAM role that your application can use when it accesses other AWS resources or specify the instance user data that can be used to perform common automated configuration tasks after an instance starts. For more information, see Create a launch template using advanced settings.
7. When you are ready to create the launch template, choose **Create launch template**.
8. To create an Auto Scaling group, choose **Create Auto Scaling group** from the confirmation page.

Change the default network interface settings (console)

Network interfaces provide connectivity to other resources in your VPC and the internet. For more information, see Provide network connectivity for your Auto Scaling instances using Amazon VPC.

This section shows you how to change the default network interface settings. For example, you can define whether you want to assign a public IPv4 address to each instance instead of defaulting to the auto-assign public IPv4 addresses setting on the subnet.

Considerations and limitations

When changing the default network interface settings, keep in mind the following considerations and limitations:

- You must configure the security groups as part of the network interface, not in the **Security groups** section of the template. You cannot specify security groups in both places.
- You cannot assign secondary private IP addresses, known as *secondary IP addresses*, to a network interface.
- If you specify an existing network interface ID, you can launch only one instance. To do this, you must use the AWS CLI or an SDK to create the Auto Scaling group. When you create the group, you must specify the Availability Zone, but not the subnet ID. Also, you can specify an existing network interface only if it has a device index of 0.
- You cannot auto-assign a public IPv4 address if you specify more than one network interface. You also cannot specify duplicate device indexes across network interfaces. Both the primary and secondary network interfaces reside in the same subnet.
- When an instance launches, a private address is automatically allocated to each network interface. The address comes from the CIDR range of the subnet in which the instance is launched. For information on specifying CIDR blocks (or IP address ranges) for your VPC or subnet, see the Amazon VPC User Guide.

To change the default network interface settings

1. Under **Network settings**, expand **Advanced network configuration**.
2. Choose **Add network interface** to configure the primary network interface, paying attention to the following fields:

    a. **Device index**: Keep the default value, 0, to apply your changes to the primary network interface (eth0).

b. **Network interface**: Keep the default value, **New interface**, to have Amazon EC2 Auto Scaling automatically create a new network interface when an instance is launched. Alternatively, you can choose an existing, available network interface with a device index of 0, but this limits your Auto Scaling group to one instance.

c. **Description**: (Optional) Enter a descriptive name.

d. **Subnet**: Keep the default **Don't include in launch template** setting.

   If the AMI specifies a subnet for the network interface, this results in an error. We recommend turning off **Auto Scaling guidance** as a workaround. After you make this change, you will not receive an error message. However, regardless of where the subnet is specified, the subnet settings of the Auto Scaling group take precedence and cannot be overridden.

e. **Auto-assign public IP**: Change whether your network interface with a device index of 0 receives a public IPv4 address. By default, instances in a default subnet receive a public IPv4 address, while instances in a nondefault subnet do not. Select **Enable** or **Disable** to override the subnet's default setting.

f. **Security groups**: Choose one or more security groups for the network interface. Each security group must be configured for the VPC that your Auto Scaling group will launch instances into. For more information, see Amazon EC2 security groups for Linux instances in the *Amazon EC2 User Guide*.

g. **Delete on termination**: Choose **Yes** to delete the network interface when the instance is terminated, or choose **No** to keep the network interface.

h. **Elastic Fabric Adapter**: To support high performance computing and machine learning use cases, change the network interface into an Elastic Fabric Adapter network interface. For more information, see Elastic Fabric Adapter in the *Amazon EC2 User Guide*.

i. **Network card index**: Choose **0** to attach the primary network interface to the network card with a device index of 0. If this option isn't available, keep the default value, **Don't include in launch template**. Attaching the network interface to a specific network card is available only for supported instance types. For more information, see Network cards in the *Amazon EC2 User Guide*.

j. **ENA Express**: For instance types that support ENA Express, choose **Enable** to enable ENA Express or **Disable** to disable it. For more information, see Improve network performance with ENA Express on Linux instances in the *Amazon EC2 User Guide*.

k. **ENA Express UDP**: If you enable **ENA Express**, you can optionally use it for UDP traffic. Choose **Enable** to enable ENA Express UDP or **Disable** to disable it.

3. To add a secondary network interface, choose **Add network interface**.

Modify the storage configuration (console)

You can modify the storage configuration for instances launched from an Amazon EBS-backed AMI or an instance store-backed AMI. You can also specify additional EBS volumes to attach to the instances. The AMI includes one or more volumes of storage, including the root volume (**Volume 1 (AMI Root)**).

To modify the storage configuration

1. In **Configure storage**, modify the size or type of volume.

   If the value you specify for volume size is outside the limits of the volume type, or smaller than the snapshot size, an error message is displayed. To help you address the issue, this message gives the minimum or maximum value that the field can accept.

   Only volumes associated with an Amazon EBS-backed AMI appear. To display information about the storage configuration for an instance launched from an instance store-backed AMI, choose **Show details** from the **Instance store volumes** section.

   To specify all EBS volume parameters, switch to the **Advanced** view in the top right corner.

2. For advanced options, expand the volume that you want to modify and configure the volume as follows:

   a. **Storage type**: The type of volume (EBS or ephemeral) to associate with your instance. The instance store (ephemeral) volume type is only available if you select an instance type that supports it. For more information, see Amazon EBS volumes in the *Amazon EBS User Guide* and Amazon EC2 instance store in the *Amazon EC2 User Guide*.

   b. **Device name**: Select from the list of available device names for the volume.

   c. **Snapshot**: Select the snapshot from which to create the volume. You can search for available shared and public snapshots by entering text into the **Snapshot** field.

   d. **Size (GiB)**: For EBS volumes, you can specify a storage size. If you have selected an AMI and instance that are eligible for the free tier, keep in mind that to stay within the free tier, you must stay under 30 GiB of total storage. For more information, see Constraints on the size and configuration of an EBS volume in the *Amazon EBS User Guide*.

e. **Volume type**: For EBS volumes, choose the volume type. For more information, see Amazon EBS volume types in the *Amazon EBS User Guide*.

f. **IOPS**: If you have selected a Provisioned IOPS SSD (`io1` and `io2`) or General Purpose SSD (gp3) volume type, then you can enter the number of I/O operations per second (IOPS) that the volume can support. This is required for io1, io2, and gp3 volumes. It is not supported for gp2, st1, sc1, or standard volumes.

g. **Delete on termination**: For EBS volumes, choose **Yes** to delete the volume when the instance is terminated, or choose **No** to keep the volume.

h. **Encrypted**: If the instance type supports EBS encryption, you can choose **Yes** to enable encryption for the volume. If you have enabled encryption by default in this Region, encryption is enabled for you. For more information, see Amazon EBS encryption and Enable encryption by default in the *Amazon EBS User Guide*.

   The default effect of setting this parameter varies with the choice of volume source, as described in the following table. In all cases, you must have permission to use the specified AWS KMS key.

   Encryption outcomes

If `Encrypted` parameter is set to...	And if source of volume is...	Then the default encryption state is...	Notes
No	New (empty) volume	Unencrypted*	N/A
	Unencrypted snapshot that you own	Unencrypted*	
	Encrypted snapshot that you own	Encrypted by same key	
	Unencrypted snapshot that is shared with you	Unencrypted*	
	Encrypted snapshot that is shared with you	Encrypted by default KMS key	

If `Encrypted` parameter is set to...	And if source of volume is...	Then the default encryption state is...	Notes
Yes	New volume	Encrypted by default KMS key	To use a non-default KMS key, specify a value for the **KMS key** parameter.
	Unencrypted snapshot that you own	Encrypted by default KMS key	
	Encrypted snapshot that you own	Encrypted by same key	
	Unencrypted snapshot that is shared with you	Encrypted by default KMS key	
	Encrypted snapshot that is shared with you	Encrypted by default KMS key	

* If encryption by default is enabled, all newly created volumes (whether or not the **Encrypted** parameter is set to **Yes**) are encrypted using the default KMS key. If you set both the **Encrypted** and **KMS key** parameters, then you can specify a non-default KMS key.

i. **KMS key**: If you chose **Yes** for **Encrypted**, then you must select a customer managed key to use to encrypt the volume. If you have enabled encryption by default in this Region, the default customer managed key is selected for you. You can select a different key or specify the ARN of any customer managed key that you previously created using the AWS Key Management Service.

3. To specify additional volumes to attach to the instances launched by this launch template, choose **Add new volume**.

Create a launch template from an existing instance (console)

To create a launch template from an existing instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. On the navigation pane, under **Instances**, choose **Instances**.
3. Select the instance and choose **Actions**, **Image and templates**, **Create template from instance**.
4. Provide a name and description.
5. Under **Auto Scaling guidance**, select the check box.
6. Adjust any settings as required, and choose **Create launch template**.
7. To create an Auto Scaling group, choose **Create Auto Scaling group** from the confirmation page.

Related resources

We provide a few JSON and YAML template snippets that you can use to understand how to declare launch templates in your AWS CloudFormation stack templates. For more information, see the AWS::EC2::LaunchTemplate and Create launch templates with AWS CloudFormation sections of the *AWS CloudFormation User Guide*.

For more information about launch templates, see Launching an instance from a launch template in the *Amazon EC2 User Guide*.

Limitations

- While you can specify a subnet in a launch template, doing so isn't necessary if you only use the launch template to create Auto Scaling groups. You can't specify the subnet for an Auto Scaling group by specifying the subnet in a launch template. The subnets for the Auto Scaling group are taken from the Auto Scaling group's own resource definition.
- For other limitations on user-defined network interfaces, see Change the default network interface settings (console).

Create a launch template using advanced settings

This topic describes how to create a launch template with advanced settings from the AWS Management Console.

To create a launch template using advanced settings

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. On the navigation pane, under **Instances**, choose **Launch Templates**, and then choose **Create launch template**.
3. Configure your launch template as described in the following topics:
   - Required settings
   - Advanced settings
4. Choose **Create launch template**.

Required settings

When you create a launch template, you must include the following required settings.

Launch template name

Enter a unique name that describes the launch template.

Application and OS Images (Amazon Machine Image)

Choose the Amazon Machine Image (AMI) that you want to use. You can either search or browse for the AMI you want to use. For best scaling efficiency, choose a custom AMI that is fully configured to launch an instance with your application code and requires few modifications on launch.

Instance type

Choose an instance type that is compatible with your AMI. You can skip adding an instance type to your launch template if you plan to use multiple instances types that are embedded in the Auto Scaling group's own resource definition. An instance type is only required if you don't plan to create a mixed instances group.

Advanced settings

The advanced settings are optional. If you do not configure any advanced settings, the specific capabilities will not be added to your instances.

Expand the **Advanced details** section to view the advanced settings. The following sections describe the most useful advanced settings to focus on when creating a launch template for an Auto Scaling group. For more information, see Advanced details in the *Amazon EC2 User Guide*.

IAM instance profile

The instance profile contains the IAM role that you want to use. When your Auto Scaling group launches an EC2 instance, the permissions defined in the associated IAM role are granted to applications running on the instance. For more information, see IAM role for applications that run on Amazon EC2 instances.

Termination protection

When enabled, this feature prevents users from terminating an instance using the Amazon EC2 console, CLI commands, and API operations. Termination protection provides an extra safeguard against accidental termination. It does not prevent Amazon EC2 Auto Scaling from terminating an instance. To control which instances Amazon EC2 Auto Scaling can terminate, see Use instance scale-in protection to control instance termination.

Detailed CloudWatch monitoring

You can enable detailed monitoring for your EC2 instances to allow them to send metric data to Amazon CloudWatch at 1-minute intervals. By default, EC2 instances send metric data to CloudWatch at 5-minute intervals. Additional charges apply. For more information, see Configure monitoring for Auto Scaling instances.

Credit specification

Amazon EC2 provides burstable performance instances, such as T2, T3, and T3a, that allow applications to burst beyond the baseline CPU performance when required. By default, these instances can burst for a limited time before their CPU usage is throttled. You can optionally enable unlimited mode so that the instances can burst beyond the baseline for as long as needed. This allows applications to sustain high CPU performance when required. Additional charges may apply. For more information, see Use an Auto Scaling group to launch a burstable performance instance as Unlimited in the *Amazon EC2 User Guide*.

Placement group name

You can specify a placement group and use a cluster or a partition strategy to influence how your instances are physically located in the AWS data center. For small Auto Scaling groups, you can also use the spread strategy. For more information, see Placement groups in the *Amazon EC2 User Guide*.

There are some considerations when using placement groups with Auto Scaling groups:

- If a placement group is specified in both the launch template and the Auto Scaling group, the placement group for the Auto Scaling group takes precedence. After the group is created, the placement group specified in the Auto Scaling group settings cannot be changed.
- In AWS CloudFormation, be careful if you define a placement group in the launch template. Amazon EC2 Auto Scaling will launch instances into the specified placement group. However, CloudFormation will not receive signals from those instances if you use an UpdatePolicy with your Auto Scaling group (though this could change in the future).

Purchasing option

You can choose **Request Spot Instances** to request Spot Instances at the Spot price, capped at the On-Demand price, and choose **Customize** to change the default Spot Instance settings. For an Auto Scaling group, you must specify a one-time request with no end date (the default). For more information, see Request Spot Instances for fault-tolerant and flexible applications. This setting may be useful in special circumstances, but in general it's best to leave it unspecified and create a mixed instances group instead. For more information, see Auto Scaling groups with multiple instance types and purchase options.

If you specify a Spot Instance request in your launch template, you can't create a mixed instances group. If you try to use a launch template that requests Spot Instances with a mixed instances group, you receive the following error message: `Incompatible launch template: You cannot use a launch template that is set to request Spot Instances (InstanceMarketOptions) when you configure an Auto Scaling group with a mixed instances policy. Add a different launch template to the group and try again.`

Capacity Reservation

Capacity Reservations allow you to reserve capacity for your Amazon EC2 instances in a specific Availability Zone for any duration. For more information, see On-Demand Capacity Reservations in the *Amazon EC2 User Guide*.

You can choose whether to launch instances into:

- any open Capacity Reservation (**Open**)
- a specific Capacity Reservation (**Target by ID**)
- a group of Capacity Reservations (**Target by group**)

To target a specific Capacity Reservation, the instance type in your launch template must match the instance type of the reservation. When you create your Auto Scaling group, use the same

Availability Zone as the Capacity Reservation. Depending on the AWS Region you choose, you can choose to target a Capacity Block instead. For more information, see Use Capacity Blocks for machine learning workloads.

To target a group of Capacity Reservations, see Reserve capacity in specific Availability Zones with Capacity Reservations . By targeting a group of Capacity Reservations, you can have capacity distributed across multiple Availability Zones to improve resiliency.

Tenancy

Amazon EC2 provides three options for the tenancy of your EC2 instances:

- Shared (**Shared**) – Multiple AWS accounts may share the same physical hardware. This is the default tenancy option when launching an instance.
- Dedicated instances (**Dedicated**) – Your instance runs on single-tenant hardware. No other AWS customer shares the same physical server. For more information, see Dedicated Instances in the *Amazon EC2 User Guide*.
- Dedicated Hosts (**Dedicated host**) – The instance runs on a physical server that is dedicated to your use. Using Dedicated Hosts makes it easier to bring your own licenses (BYOL) that have dedicated hardware requirements to EC2 and meet compliance use cases. If you choose this option, you must provide a host resource group for **Tenancy host resource group**. For more information, see Dedicated Hosts in the *Amazon EC2 User Guide*.

Support for Dedicated Hosts is only available if you specify a host resource group. You can't target a specific host ID or use host placement affinity.

- If you try to use a launch template that specifies a host ID, you receive the following error message: `Incompatible launch template: Tenancy host ID is not supported for Auto Scaling.`
- If you try to use a launch template that specifies host placement affinity, you receive the following error message: `Incompatible launch template: Auto Scaling does not support host placement affinity.`

Tenancy host resource group

With AWS License Manager, you can bring your own licenses to AWS and manage them centrally. A host resource group is a group of Dedicated Hosts that are linked to a specific License Manager license configuration. Host resource groups allow you to easily launch EC2 instances onto Dedicated Hosts that match your software licensing needs. You do not need to manually allocate Dedicated Hosts ahead of time. They are automatically created as needed.

Note that when you associate an AMI with a license configuration, that AMI can only be associated with one host resource group at a time. For more information, see Host resource groups in AWS License Manager in the *License Manager User Guide*.

License configurations

With this setting, you can specify a license configuration for your instances without restricting their tenancy to Dedicated Hosts. The license configuration tracks the software licenses deployed on the instances so you can monitor your license usage and compliance. For more information, see Create a self-managed license in the *License Manager User Guide*.

Metadata accessible

You can choose whether to enable or disable access to the HTTP endpoint of the instance metadata service. By default, the HTTP endpoint is enabled. If you choose to disable the endpoint, access to your instance metadata is turned off. You can specify the condition to require IMDSv2 only when the HTTP endpoint is enabled. For more information, see Configure the instance metadata options in the *Amazon EC2 User Guide*.

Metadata version

You can choose to require the use of Instance Metadata Service Version 2 (IMDSv2) when requesting instance metadata. If you do not specify a value, the default is to support both IMDSv1 and IMDSv2. For more information, see Configure the instance metadata options in the *Amazon EC2 User Guide*.

Metadata token response hop limit

You can set the allowable number of network hops for the metadata token. If you do not specify a value, the default is 1. For more information, see Configure the instance metadata options in the *Amazon EC2 User Guide*.

User data

You can customize and finish configuring your instances at launch time by specifying shell scripts or cloud-init directives as user data. The user data runs when the instance initially starts up, allowing you to automatically install applications, dependencies, or customizations at launch time. For more information, see Run commands on your Linux instance at launch in the *Amazon EC2 User Guide*.

If you have large downloads or complex scripts, this adds to the time it takes for the instance to become ready for use. In which case, you may need to configure a lifecycle hook to delay an instance from reaching the `InService` state until it's fully provisioned. For more information

about adding a lifecycle hook to your Auto Scaling group, see Amazon EC2 Auto Scaling lifecycle hooks.

Request Spot Instances for fault-tolerant and flexible applications

In your launch template, you can optionally request Spot Instances with no end date or duration. Amazon EC2 Spot Instances are spare capacity available at steep discounts compared to the EC2 On-Demand price. Spot Instances are a cost-effective choice if you can be flexible about when your applications run and if your applications can be interrupted. For more information about creating a launch template that requests Spot Instances, see Create a launch template using advanced settings.

> **Important**
>
> Spot Instances are typically used to supplement On-Demand Instances. For this scenario, you can specify the same settings that are used to launch Spot Instances as part of the settings of your Auto Scaling group. When you specify the settings as part of the Auto Scaling group, you can request to launch Spot Instances only after launching a certain number of On-Demand Instances and then continue to launch some combination of On-Demand Instances and Spot Instances as the group scales. For more information, see Auto Scaling groups with multiple instance types and purchase options.

This topic describes how to launch only Spot Instances in your Auto Scaling group by specifying settings in a launch template, rather than in the Auto Scaling group itself. The information in this topic also applies to Auto Scaling groups that request Spot Instances with a launch configuration. The difference is that a launch configuration requires a maximum price, but for launch templates, the maximum price is optional.

When you create a launch template to launch only Spot Instances, keep the following considerations in mind:

- **Spot price.** You pay only the current Spot price for the Spot Instances that you launch. This pricing changes slowly over time based on long-term trends in supply and demand. For more information, see Spot Instances and Pricing and savings in the *Amazon EC2 User Guide*.
- **Setting your maximum price.** You can optionally include a maximum price per hour for Spot Instances in your launch template. If your maximum price exceeds the current Spot price, the

Amazon EC2 Spot service fulfills your request immediately if capacity is available. If the price for Spot Instances rises above your maximum price for a running instance in your Auto Scaling group, it terminates your instance.

> **Warning**
>
> Your application might not run if you do not receive any Spot Instances, such as when your maximum price is too low. To take advantage of the Spot Instances available for as long as possible, set your maximum price close to the On-Demand price.

- **Balancing across Availability Zones.** If you specify multiple Availability Zones, Amazon EC2 Auto Scaling distributes the Spot requests across the specified zones. If your maximum price is too low in one Availability Zone for any requests to be fulfilled, Amazon EC2 Auto Scaling checks whether requests were fulfilled in the other zones. If so, Amazon EC2 Auto Scaling cancels the requests that failed and redistributes them across the Availability Zones that have requests fulfilled. If the price in an Availability Zone with no fulfilled requests drops enough that future requests succeed, Amazon EC2 Auto Scaling rebalances across all of the Availability Zones.
- **Spot Instance termination.** Spot Instances can be terminated at any time. The Amazon EC2 Spot service can terminate Spot Instances in your Auto Scaling group as the availability of, or price for, Spot Instances changes. When scaling or performing health checks, Amazon EC2 Auto Scaling can also terminate Spot Instances in the same way that it can terminate On-Demand Instances. When an instance is terminated, any storage is deleted.
- **Maintaining your desired capacity.** When a Spot Instance is terminated, Amazon EC2 Auto Scaling attempts to launch another Spot Instance to maintain the desired capacity for the group. If the current Spot price is less than your maximum price, it launches a Spot Instance. If the request for a Spot Instance is unsuccessful, it keeps trying.
- **Changing your maximum price.** To change your maximum price, create a new launch template or update an existing launch template with the new maximum price, and then associate it with your Auto Scaling group. The existing Spot Instances continue to run as long as the maximum price specified in the launch template used for those instances is higher than the current Spot price. If you did not set a maximum price, the default maximum price is the On-Demand price.

Use Capacity Blocks for machine learning workloads

Capacity Blocks help you reserve highly sought-after GPU instances on a future date to support your short-duration, machine learning (ML) workloads.

For an overview of Capacity Blocks and how they work, see Capacity Blocks for ML in the *Amazon EC2 User Guide*.

To start using Capacity Blocks, you create a capacity reservation in a specific Availability Zone. Capacity Blocks are delivered as `targeted` capacity reservations in a single Availability Zone. When you create your launch template, specify the Capacity Block's reservation ID and instance type. Then, update your Auto Scaling group to use the launch template you created and the Capacity Block's Availability Zone. When your Capacity Block reservation begins, use scheduled scaling to launch the same number of instances as your Capacity Block reservation.

> **Important**
>
> Capacity Blocks are only available for certain Amazon EC2 instance types and AWS Regions. For more information, see Prerequisites in the *Amazon EC2 User Guide*.

Contents

- Operational guidelines
- Specify a Capacity Block in your launch template
- Limitations
- Related resources

Operational guidelines

The following are basic operational guidelines that you should follow when using a Capacity Block with an Auto Scaling group.

- Scale in your Auto Scaling group to zero more than 30 minutes before the Capacity Block reservation end time. Amazon EC2 will terminate any instances that are still running 30 minutes before the end time of the Capacity Block.
- We recommend that you use scheduled scaling to scale out (add instances) and scale in (remove instances) at the appropriate reservation times. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling.
- Add lifecycle hooks as needed to perform a graceful shutdown of your application inside the instances when scaling in. Leave enough time for the lifecycle action to complete *before* Amazon EC2 starts forcibly terminating your instances 30 minutes before the Capacity Block reservation end time. For more information, see Amazon EC2 Auto Scaling lifecycle hooks.

- Make sure that the Auto Scaling group points to the correct version of the launch template for the entire duration of the reservation. We recommend pointing to a specific version of the launch template instead of the `$Default` or `$Latest` version.

> **Note**
>
> If you leave a Capacity Block instance running until the end of the reservation and Amazon EC2 reclaims it, the scaling activities for your Auto Scaling group state that it was "`taken out of service in response to an EC2 health check that indicated it had been terminated or stopped`", even though it was purposely reclaimed at the end of the Capacity Block. Similarly, Amazon EC2 Auto Scaling will attempt to replace the instance in the same manner as it does for any instance that fails a health check. For more information, see Health checks for instances in an Auto Scaling group.

Specify a Capacity Block in your launch template

To create a launch template that targets a specific Capacity Block for your Auto Scaling group, use one of the following methods:

Console

To specify a Capacity Block in your launch template (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the top navigation bar, select the AWS Region where you created your Capacity Block.
3. On the navigation pane, under **Instances**, choose **Launch Templates**.
4. Choose **Create launch template**, and create the launch template. Include the ID of the Amazon Machine Image (AMI), the instance type, and any other launch template settings as needed.
5. Expand the **Advanced details** section to view the advanced settings.
6. For **Purchasing option**, choose **Capacity Blocks**.
7. For **Capacity reservation**, choose **Target by ID**, and then for **Capacity reservation - Target by ID**, choose the capacity reservation ID of an existing Capacity Block.
8. When you have finished, choose **Create launch template**.

For help creating an Auto Scaling group with a launch template, see Create an Auto Scaling group using a launch template.

AWS CLI

To specify a Capacity Block in your launch template (AWS CLI)

Use the following create-launch-template command to create a launch template that specifies an existing Capacity Block reservation ID. Replace each *user input placeholder* with your own information.

```
aws ec2 create-launch-template --launch-template-name my-template-for-capacity-block
  \
  --version-description AutoScalingVersion1 --region us-east-2 \
  --launch-template-data file://config.json
```

> **Tip**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Contents of `config.json`.

```
{
    "ImageId": "ami-04d5cc9b88example",
    "InstanceType": "p4d.24xlarge",
    "SecurityGroupIds": [
        "sg-903004f88example"
    ],
    "KeyName": "MyKeyPair",
    "InstanceMarketOptions": {
        "MarketType": "capacity-block"
    },
    "CapacityReservationSpecification": {
        "CapacityReservationTarget": {
            "CapacityReservationId": "cr-02168da1478b509e0"
        }
    }
}
```

The following is example output.

```
{
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-068f72b724example",
        "LaunchTemplateName": "my-template-for-capacity-block",
        "CreateTime": "2023-10-27T15:12:44.000Z",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
    }
}
```

You can use the following describe-launch-template-versions command to verify the Capacity Block reservation ID associated with the launch template.

```
aws ec2 describe-launch-template-versions --launch-template-names my-template-for-capacity-block \
  --region us-east-2
```

The following is example output for a launch template that specifies a Capacity Block reservation.

```
{
    "LaunchTemplateVersions": [
        {
            "LaunchTemplateId": "lt-068f72b724example",
            "LaunchTemplateName": "my-template-for-capacity-block",
            "VersionNumber": 1,
            "CreateTime": "2023-10-27T15:12:44.000Z",
            "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
            "DefaultVersion": true,
            "LaunchTemplateData": {
                "ImageId": "ami-04d5cc9b88example",
                "InstanceType": "p5.48xlarge",
                "SecurityGroupIds": [
                    "sg-903004f88example"
                ],
                "KeyName": "MyKeyPair",
                "InstanceMarketOptions": {
                    "MarketType": "capacity-block"
                },
```

```
                "CapacityReservationSpecification": {
                    "CapacityReservationTarget": {
                        "CapacityReservationId": "cr-02168da1478b509e0"
                    }
                }
            }
        }
    ]
}
```

Limitations

- Support for Capacity Blocks is only available if your Auto Scaling group has a compatible configuration. Mixed instances groups and warm pools are not supported.
- You can only target one Capacity Block at a time.

Related resources

- For the prerequisites and recommendations for using P5 Instances, see Get started with P5 instances in the *Amazon EC2 User Guide*.
- Amazon EKS supports using Capacity Blocks to support your short duration, machine learning (ML) workloads on Amazon EKS clusters. For more information, see Capacity Blocks for ML in the Amazon EKS User Guide.
- You can use Capacity Blocks with supported instance types and Regions. However, On-Demand Capacity Reservations provide flexibility to reserve capacity for other instances types and Regions. For a tutorial that shows you how to use the On-Demand Capacity Reservation option, see Reserve capacity in specific Availability Zones with Capacity Reservations .

Migrate your Auto Scaling groups to launch templates

Starting in 2023, you can not call `CreateLaunchConfiguration` with new Amazon EC2 instance types released after December 31, 2022. For more information, see Auto Scaling launch configurations.

To migrate your Auto Scaling groups from launch configurations to launch templates, see the following steps.

> **Important**
>
> Before you continue, confirm that you have the permissions required to work with launch templates. For more information, see Permissions to work with launch templates.

Step 1: Find Auto Scaling groups that use launch configurations

To identify whether you have Auto Scaling groups that are still using launch configurations, run the following describe-auto-scaling-groups command using the AWS CLI. Replace *REGION* with your AWS Region.

```
aws autoscaling describe-auto-scaling-groups --region REGION \
  --query 'AutoScalingGroups[?LaunchConfigurationName!=`null`]'
```

The following is example output.

```
[
    {
 "AutoScalingGroupName": "group-1",
 "AutoScalingGroupARN": "arn",
 "LaunchConfigurationName": "my-launch-config",
 "MinSize": 1,
 "MaxSize": 5,
 "DesiredCapacity": 2,
 "DefaultCooldown": 300,
 "AvailabilityZones": [
            "us-west-2a",
            "us-west-2b",
            "us-west-2c"
        ],
 "LoadBalancerNames": [],
 "TargetGroupARNs": [],
 "HealthCheckType": "EC2",
 "HealthCheckGracePeriod": 300,
 "Instances": [
            {
                "ProtectedFromScaleIn": false,
                "AvailabilityZone": "us-west-2a",
                "LaunchConfigurationName": "my-launch-config",
                "InstanceId": "i-05b4f7d5be44822a6",
```

```
                "InstanceType": "t3.micro",
                "HealthStatus": "Healthy",
                "LifecycleState": "InService"
            },
            {
                "ProtectedFromScaleIn": false,
                "AvailabilityZone": "us-west-2b",
                "LaunchConfigurationName": "my-launch-config",
                "InstanceId": "i-0c20ac468fa3049e8",
                "InstanceType": "t3.micro",
                "HealthStatus": "Healthy",
                "LifecycleState": "InService"
            }
 ],
 "CreatedTime": "2023-03-09T22:15:11.611Z",
 "SuspendedProcesses": [],
 "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782",
 "EnabledMetrics": [],
 "Tags": [
            {
  "ResourceId": "group-1",
  "ResourceType": "auto-scaling-group",
  "Key": "environment",
  "Value": "production",
  "PropagateAtLaunch": true
            }
        ],
 "TerminationPolicies": [
     "Default"
 ],
 "NewInstancesProtectedFromScaleIn": false,
 "ServiceLinkedRoleARN": "arn",
       "TrafficSources": []
    },

    ... additional groups ...

]
```

Alternatively, to remove everything except the Auto Scaling group names with the names of their respective launch configurations and tags in the output, run the following command:

```
aws autoscaling describe-auto-scaling-groups --region REGION \
```

```
 --query 'AutoScalingGroups[?LaunchConfigurationName!=`null`].{AutoScalingGroupName:
AutoScalingGroupName, LaunchConfigurationName: LaunchConfigurationName, Tags: Tags}'
```

The following shows example output.

```
[
    {
        "AutoScalingGroupName": "group-1",
        "LaunchConfigurationName": "my-launch-config",
        "Tags": [
            {
                "ResourceId": "group-1",
                "ResourceType": "auto-scaling-group",
                "Key": "environment",
                "Value": "production",
                "PropagateAtLaunch": true
            }
        ]
    },

    ... additional groups ...

]
```

For more information about filtering, see Filtering AWS CLI output in the *AWS Command Line Interface User Guide*.

Step 2: Copy a launch configuration to a launch template

You can copy a launch configuration to a launch template using the following procedure. Then, you can add it to your Auto Scaling group.

Copying multiple launch configurations results in identically named launch templates. To change the name given to a launch template during the copying process, you must copy the launch configurations one by one.

> **Note**
>
> The copying feature is available only from the console.

To copy a launch configuration to a launch template (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the left navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Choose **Launch configurations** near the top of the page. When prompted for confirmation, choose **View launch configurations** to confirm that you want to view the **Launch configurations** page.
4. Select the launch configuration you want to copy and choose **Copy to launch template, Copy selected**. This sets up a new launch template with the same name and options as the launch configuration that you selected.
5. For **New launch template name**, you can use the name of the launch configuration (the default) or enter a new name. Launch template names must be unique.
6. (Optional) Select **Create an Auto Scaling group using the new template**.

   You can skip this step to finish copying the launch configuration. You do not need to create a new Auto Scaling group.
7. Choose **Copy**.

To copy all launch configurations to launch templates (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Auto Scaling**, choose **Launch Configurations**.
3. Choose **Copy to launch template, Copy all**. This copies each launch configuration in the current Region to a new launch template with the same name and options.
4. Choose **Copy**.

Step 3: Update an Auto Scaling group to use a launch template

After creating a launch template, you're ready to add it to your Auto Scaling group.

To update an Auto Scaling group to use a launch template (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

A split pane opens up in the bottom part of the page, showing information about the group that's selected.

3. On the **Details** tab, choose **Launch configuration**, **Edit**.
4. Choose **Switch to launch template**.
5. For **Launch template**, select your launch template.
6. For **Version**, select the launch template version, as needed. After you create versions of a launch template, you can choose whether the Auto Scaling group uses the default or the latest version of the launch template when scaling out.
7. Choose **Update**.

To update an Auto Scaling group to use a launch template (AWS CLI)

The following update-auto-scaling-group command updates the specified Auto Scaling group to use the initial version of the specified launch template.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-template LaunchTemplateName=my-template-for-auto-scaling,Version='1'
```

For more examples of using CLI commands to update an Auto Scaling group to use a launch template, see Update an Auto Scaling group to use a launch template.

Step 4: Replace your instances

After you replace the launch configuration with a launch template, any new instances will use the new launch template. Existing instances are not affected.

To update the existing instances, you can start an instance refresh to replace the instances in your Auto Scaling group instead of manually replacing instances a few at a time. For more information, see Use an instance refresh to update instances in an Auto Scaling group. If the group is large, an instance refresh can be particularly helpful.

Alternatively, you can allow automatic scaling to gradually replace existing instances with new instances based on the group's termination policies, or you can terminate them. Manual termination forces your Auto Scaling group to launch new instances to maintain the group's desired capacity. For more information, see Terminate an instance in the *Amazon EC2 User Guide*.

Additional information

For more information, see Amazon EC2 Auto Scaling will no longer add support for new EC2 features to Launch Configurations on the AWS Compute Blog.

For a topic that takes you through how to migrate AWS CloudFormation stacks from launch configurations to launch templates, see Migrate AWS CloudFormation stacks to launch templates.

Migrate AWS CloudFormation stacks to launch templates

You can migrate your existing AWS CloudFormation stack templates from launch configurations to launch templates. To do this, add a launch template directly to an existing stack template and then associate the launch template with the Auto Scaling group in the stack template. Then, use your modified template to update your stack.

When migrating to launch templates, this topic saves you time by providing instructions for rewriting the launch configurations in your CloudFormation stack templates as launch templates. For more information about migrating launch configurations to launch templates, see Migrate your Auto Scaling groups to launch templates.

Topics

- Find Auto Scaling groups that use a launch configuration
- Update a stack to use a launch template
- Understand update behavior of stack resources
- Track the migration
- Launch configuration mapping reference

Find Auto Scaling groups that use a launch configuration

To find Auto Scaling groups that use a launch configuration

- Use the following describe-auto-scaling-groups command to list the names of Auto Scaling groups that are using launch configurations in the specified Region. Include the `--filters` option to narrow the results to groups associated with a CloudFormation stack (by filtering by the `aws:cloudformation:stack-name` tag key).

```
aws autoscaling describe-auto-scaling-groups --region REGION \
```

```
  --filters Name=tag-key,Values=aws:cloudformation:stack-name \
  --query 'AutoScalingGroups[?LaunchConfigurationName!
=`null`].AutoScalingGroupName'
```

The following shows example output.

```
[
    "{stack-name}-group-1",
    "{stack-name}-group-2",
    "{stack-name}-group-3"
]
```

You can find additional useful AWS CLI commands for finding Auto Scaling groups to migrate and filtering the output in Migrate your Auto Scaling groups to launch templates.

> **Important**
>
> If your stack resources have AWSEB in their name, this means they were created through AWS Elastic Beanstalk. In this case, you must update the Beanstalk environment to direct Elastic Beanstalk to remove the launch configuration and replace it with a launch template.

Update a stack to use a launch template

Follow the steps in this section to do the following:

- Rewrite the launch configuration as a launch template using the equivalent launch template properties.
- Associate the new launch template with the Auto Scaling group.
- Deploy these updates.

To modify the stack template and update the stack

1. Follow the same general procedures for modifying the stack template described in Modifying a stack template in the *AWS CloudFormation User Guide*.
2. Rewrite the launch configuration as a launch template. See the following example:

   Example: A simple launch configuration

```
---
Resources:
  myLaunchConfig:
    Type: AWS::AutoScaling::LaunchConfiguration
    Properties:
      ImageId: ami-02354e95b3example
      InstanceType: t3.micro
      SecurityGroups:
        - !Ref EC2SecurityGroup
      KeyName: MyKeyPair
      BlockDeviceMappings:
        - DeviceName: /dev/xvda
          Ebs:
            VolumeSize: 150
            DeleteOnTermination: true
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash -xe
          yum install -y aws-cfn-bootstrap
          /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} --resource myASG
 --region ${AWS::Region}
```

Example: The launch template equivalent

```
---
Resources:
  myLaunchTemplate:
    Type: AWS::EC2::LaunchTemplate
    Properties:
      LaunchTemplateName: !Sub ${AWS::StackName}-launch-template
      LaunchTemplateData:
        ImageId: ami-02354e95b3example
        InstanceType: t3.micro
        SecurityGroupIds:
          - Ref! EC2SecurityGroup
        KeyName: MyKeyPair
        BlockDeviceMappings:
          - DeviceName: /dev/xvda
            Ebs:
              VolumeSize: 150
              DeleteOnTermination: true
        UserData:
```

```
          Fn::Base64: !Sub |
            #!/bin/bash -x
            yum install -y aws-cfn-bootstrap
            /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} --resource
myASG --region ${AWS::Region}
```

For reference information for all the properties that Amazon EC2 supports, see AWS::EC2::LaunchTemplate in the *AWS CloudFormation User Guide*.

Note how the launch template includes the `LaunchTemplateName` property with a value of `!Sub ${AWS::StackName}-launch-template`. This is required if you want the name of the launch template to include the stack name.

3. If the **`IamInstanceProfile`** property is present in your launch configuration, you must convert it to a structure and specify either the name or the ARN of the instance profile. For an example, see AWS::EC2::LaunchTemplate.
4. If the **`AssociatePublicIpAddress`**, **`InstanceMonitoring`**, or **`PlacementTenancy`** properties are present in your launch configuration, you must convert these to a structure. For examples, see AWS::EC2::LaunchTemplate.

   An exception is when the value for the `MapPublicIpOnLaunch` property on the subnets you used for your Auto Scaling group matches the value for the `AssociatePublicIpAddress` property in your launch configuration. In this case, you can ignore the `AssociatePublicIpAddress` property. The `AssociatePublicIpAddress` property is only used to override the `MapPublicIpOnLaunch` property to change whether instances receive a public IPv4 address at launch.
5. You can copy security groups from the **`SecurityGroups`** property to one of two places in your launch template. Normally, you copy the security groups to the `SecurityGroupIds` property. However, if you create a `NetworkInterfaces` structure within your launch template to specify the `AssociatePublicIpAddress` property, then you must copy the security groups to the `Groups` property of the network interface instead.
6. If any `BlockDeviceMapping` structures are present in your launch configuration with **`NoDevice`** set to `true`, then you must specify an empty string for `NoDevice` in your launch template to have Amazon EC2 omit the device.
7. If the **`SpotPrice`** property is present in your launch configuration, we recommend that you omit it from your launch template. Your Spot Instances will launch at the current Spot price. This price will never exceed the On-Demand price.

To request Spot Instances, you have two mutually exclusive options:

- The first is to use the `InstanceMarketOptions` structure in your launch template (not recommended). For more information, see AWS::EC2::LaunchTemplate InstanceMarketOptions in the *AWS CloudFormation User Guide*.
- The other is to add a `MixedInstancesPolicy` structure to your Auto Scaling group. Doing so provides you with more options for how you make the request. A Spot Instance request in your launch template doesn't support more than one instance type selection per Auto Scaling group. However, a mixed instances policy does support more than one instance type selection per Auto Scaling group. Spot Instance requests benefit from having more than one instance type to choose from. For more information, see AWS::AutoScaling::AutoScalingGroup MixedInstancesPolicy in the *AWS CloudFormation User Guide*.

8. Remove the **`LaunchConfigurationName`** property from the AWS::AutoScaling::AutoScalingGroup resource. Add the launch template in its place.

   In the following examples, the Ref intrinsic function gets the ID of the AWS::EC2::LaunchTemplate resource with the logical ID `myLaunchTemplate`. The GetAtt function gets the latest version number (for example, 1) of the launch template for the `Version` property.

   Example: Without a mixed instances policy

```
---
Resources:
  myASG:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      LaunchTemplate:
        LaunchTemplateId: !Ref myLaunchTemplate
        Version: !GetAtt myLaunchTemplate.LatestVersionNumber
      ...
```

   Example: With a mixed instances policy

```
---
Resources:
  myASG:
```

```
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      MixedInstancesPolicy:
        LaunchTemplate:
          LaunchTemplateSpecification:
            LaunchTemplateId: !Ref myLaunchTemplate
            Version: !GetAtt myLaunchTemplate.LatestVersionNumber
      ...
```

For reference information for all the properties that Amazon EC2 Auto Scaling supports, see AWS::AutoScaling::AutoScalingGroup in the *AWS CloudFormation User Guide*.

9. When you are ready to deploy these updates, follow the CloudFormation procedures to update the stack with your modified stack template. For more information, see Modifying a stack template in the *AWS CloudFormation User Guide*.

Understand update behavior of stack resources

CloudFormation updates stack resources by comparing changes between the updated template you provide, and resource configurations you described in the previous version of your stack template. Resource configurations that haven't changed remain unaffected during the update process.

CloudFormation supports the UpdatePolicy attribute for Auto Scaling groups. During an update, if `UpdatePolicy` is set to `AutoScalingRollingUpdate`, CloudFormation replaces `InService` instances after you perform the steps in this procedure. If `UpdatePolicy` is set to `AutoScalingReplacingUpdate`, CloudFormation replaces the Auto Scaling group and its warm pool (if one exists).

If you didn't specify an `UpdatePolicy` attribute for your Auto Scaling group, the launch template is checked for correctness, but CloudFormation does not deploy any change across the instances in the Auto Scaling group. All new instances will use your launch template, but existing instances continue to run with the launch configuration that they were originally launched with (despite the launch configuration not existing). The exception is when you change your purchase options, for example, by adding a mixed instances policy. In this case, your Auto Scaling group gradually replaces the existing instances with new instances to match the new purchase options.

If you have to roll back a change to move from launch configurations to launch templates, make sure to test the roll back operation.

Track the migration

To track the migration

1. In the AWS CloudFormation console, select the stack that you updated and then choose the **Events** tab to view the stacks events.
2. To update the event list with the most recent events, choose the refresh button in the CloudFormation console.
3. While your stack is updating, you will notice multiple events for each resource update. If you see an exception in the **Status reason** column that indicates a problem when trying to create the launch template, see Troubleshoot Amazon EC2 Auto Scaling: Launch templates for potential causes.
4. (Optional) Depending on your use of the `UpdatePolicy` attribute, you can monitor the progress of your Auto Scaling group from the Auto Scaling groups page of the Amazon EC2 console. Select the Auto Scaling group. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched or terminated instances, or whether the scaling activity is still in progress.
5. When the stack update is complete, CloudFormation issues an `UPDATE_COMPLETE` stack event. For more information, see Monitoring the progress of a stack update in the *AWS CloudFormation User Guide*.
6. After the stack update is complete, open the Launch templates page and Launch configurations page of the Amazon EC2 console. You will notice a new launch template is created, and the launch configuration is deleted.

Launch configuration mapping reference

For reference purposes, the following table lists all the top-level properties in the AWS::AutoScaling::LaunchConfiguration resource with their corresponding property in the AWS::EC2::LaunchTemplate resource.

Launch configuration source property	Launch template target property
`AssociatePublicIpAddress`	`NetworkInterfaces.AssociatePublicIpAddress`
`BlockDeviceMappings`	`BlockDeviceMappings`

Launch configuration source property	Launch template target property
`ClassicLinkVPCId`	Not available[1]
`ClassicLinkVPCSecurityGroups`	Not available[1]
`EbsOptimized`	`EbsOptimized`
`IamInstanceProfile`	Either `IamInstanceProfile.Arn` or `IamInstanceProfile.Name`, but not both
`ImageId`	`ImageId`
`InstanceId`	`InstanceId`
`InstanceMonitoring`	`Monitoring.Enabled`
`InstanceType`	`InstanceType`
`KernelId`	`KernelId`
`KeyName`	`KeyName`
`LaunchConfigurationName`	`LaunchTemplateName`
`MetadataOptions`	`MetadataOptions`
`PlacementTenancy`	`Placement.Tenancy`
`RamDiskId`	`RamDiskId`
`SecurityGroups`	Either `SecurityGroupIds` or `NetworkInterfaces.Groups`, but not both
`SpotPrice`	`InstanceMarketOptions.SpotOptions.MaxPrice`
`UserData`	`UserData`

[1] The `ClassicLinkVPCId` and `ClassicLinkVPCSecurityGroups` properties are not available to use in a launch template because EC2-Classic is no longer available.

Examples for creating and managing launch templates with the AWS CLI

You can create and manage launch templates through the AWS Management Console, AWS Command Line Interface (AWS CLI), or SDKs. This section shows you examples of creating and managing launch templates for Amazon EC2 Auto Scaling from the AWS CLI.

Contents

- Example usage
- Create a basic launch template
- Specify tags that tag instances at launch
- Specify an IAM role to pass to instances
- Assign public IP addresses
- Specify a user data script that configures instances at launch
- Specify a block device mapping
- Specify Dedicated Hosts to bring software licenses from external vendors
- Specify an existing network interface
- Create multiple network interfaces
- Manage your launch templates
- Update an Auto Scaling group to use a launch template

Example usage

```
{
    "LaunchTemplateName": "my-template-for-auto-scaling",
    "VersionDescription": "test description",
    "LaunchTemplateData": {
        "ImageId": "ami-04d5cc9b88example",
        "InstanceType": "t2.micro",
        "SecurityGroupIds": [
```

```
            "sg-903004f88example"
        ],
        "KeyName": "MyKeyPair",
        "Monitoring": {
            "Enabled": true
        },
        "Placement": {
            "Tenancy": "dedicated"
        },
        "CreditSpecification": {
            "CpuCredits": "unlimited"
        },
        "MetadataOptions": {
            "HttpTokens": "required",
            "HttpPutResponseHopLimit": 1,
            "HttpEndpoint": "enabled"
        }
    }
}
```

Create a basic launch template

To create a basic launch template, use the create-launch-template command as follows, with these modifications:

- Replace `ami-04d5cc9b88example` with the ID of the AMI from which to launch the instances.
- Replace `t2.micro` with an instance type that is compatible with the AMI that you specified.

This example creates a launch template with the name *`my-template-for-auto-scaling`*. If the instances created by this launch template are launched in a default VPC, they receive a public IP address by default. If the instances are launched in a nondefault VPC, they do not receive a public IP address by default.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data
 '{"ImageId":"ami-04d5cc9b88example","InstanceType":"t2.micro"}'
```

For more information about quoting JSON-formatted parameters, see Using quotation marks with strings in the AWS CLI in the *AWS Command Line Interface User Guide*.

Alternatively, you can specify the JSON-formatted parameters in a configuration file.

The following example creates a basic launch template, referencing a configuration file for launch template parameter values.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data file://config.json
```

Contents of `config.json`:

```
{
    "ImageId":"ami-04d5cc9b88example",
    "InstanceType":"t2.micro"
}
```

Specify tags that tag instances at launch

The following example adds a tag (for example, `purpose=webserver`) to instances at launch.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data '{"TagSpecifications":[{"ResourceType":"instance","Tags":
[{"Key":"purpose","Value":"webserver"}]}],"ImageId":"ami-04d5cc9b88example","InstanceType":"t2.
```

> **Note**
>
> If you specify instance tags in your launch template and then you choose to propagate your Auto Scaling group's tags to its instances, all the tags are merged. If the same tag key is specified for a tag in your launch template and a tag in your Auto Scaling group, then the tag value from the group takes precedence.

Specify an IAM role to pass to instances

The following example specifies the name of the instance profile associated with the IAM role to pass to instances at launch. For more information, see IAM role for applications that run on Amazon EC2 instances.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
--launch-template-data '{"IamInstanceProfile":{"Name":"my-instance-
profile"},"ImageId":"ami-04d5cc9b88example","InstanceType":"t2.micro"}'
```

Assign public IP addresses

The following create-launch-template example configures the launch template to assign public addresses to instances launched in a nondefault VPC.

> **Note**
>
> When you specify a network interface, specify a value for `Groups` that corresponds to security groups for the VPC that your Auto Scaling group will launch instances into. Specify the VPC subnets as properties of the Auto Scaling group.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data '{"NetworkInterfaces":
[{"DeviceIndex":0,"AssociatePublicIpAddress":true,"Groups":
["sg-903004f88example"],"DeleteOnTermination":true}],"ImageId":"ami-04d5cc9b88example","Instanc
```

Specify a user data script that configures instances at launch

The following example specifies a user data script as a base64-encoded string that configures instances at launch. The create-launch-template command requires base64-encoded user data.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
--launch-template-data
 '{"UserData":"IyEvYmluL2Jhc...","ImageId":"ami-04d5cc9b88example","InstanceType":"t2.micro"}'
```

Specify a block device mapping

The following create-launch-template example creates a launch template with a block device mapping: a 22-gigabyte EBS volume mapped to `/dev/xvdcz`. The `/dev/xvdcz` volume uses the

General Purpose SSD (gp2) volume type and is deleted when terminating the instance it is attached to.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data '{"BlockDeviceMappings":[{"DeviceName":"/dev/xvdcz","Ebs":
{"VolumeSize":22,"VolumeType":"gp2","DeleteOnTermination":true}}],"ImageId":"ami-04d5cc9b88exam
```

Specify Dedicated Hosts to bring software licenses from external vendors

If you specify *host* tenancy, you can specify a host resource group and a License Manager license configuration to bring eligible software licenses from external vendors. Then, you can use the licenses on EC2 instances by using the following create-launch-template command.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data '{"Placement":
{"Tenancy":"host","HostResourceGroupArn":"arn"},"LicenseSpecifications":
[{"LicenseConfigurationArn":"arn"}],"ImageId":"ami-04d5cc9b88example","InstanceType":"t2.micro"
```

Specify an existing network interface

The following create-launch-template example configures the primary network interface to use an existing network interface.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data '{"NetworkInterfaces":
[{"DeviceIndex":0,"NetworkInterfaceId":"eni-
b9a5ac93","DeleteOnTermination":false}],"ImageId":"ami-04d5cc9b88example","InstanceType":"t2.mi
```

Create multiple network interfaces

The following create-launch-template example adds a secondary network interface. The primary network interface has a device index of 0, and the secondary network interface has a device index of 1.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
```

```
  --launch-template-data '{"NetworkInterfaces":[{"DeviceIndex":0,"Groups":
["sg-903004f88example"],"DeleteOnTermination":true},{"DeviceIndex":1,"Groups":
["sg-903004f88example"],"DeleteOnTermination":true}],"ImageId":"ami-04d5cc9b88example","Instanc
```

If you use an instance type that supports multiple network cards and Elastic Fabric Adapters (EFAs), you can add a secondary interface to a secondary network card and enable EFA by using the following create-launch-template command. For more information, see Adding an EFA to a launch template in the *Amazon EC2 User Guide*.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling --
version-description version1 \
  --launch-template-data '{"NetworkInterfaces":
[{"NetworkCardIndex":0,"DeviceIndex":0,"Groups":
["sg-7c2270198example"],"InterfaceType":"efa","DeleteOnTermination":true},
{"NetworkCardIndex":1,"DeviceIndex":1,"Groups":
["sg-7c2270198example"],"InterfaceType":"efa","DeleteOnTermination":true}],"ImageId":"ami-09d95
```

Warning

The p4d.24xlarge instance type incurs higher costs than the other examples in this section. For more information about pricing for P4d instances, see Amazon EC2 P4d Instances pricing.

Note

Attaching multiple network interfaces from the same subnet to an instance can introduce asymmetric routing, especially on instances using a variant of non-Amazon Linux. If you need this type of configuration, you must configure the secondary network interface within the OS. For an example, see How can I make my secondary network interface work in my Ubuntu EC2 instance? in the AWS Knowledge Center.

Manage your launch templates

The AWS CLI includes several other commands that help you manage your launch templates.

Contents

- List and describe your launch templates

- Create a launch template version
- Delete a launch template version
- Delete a launch template

List and describe your launch templates

You can use two AWS CLI commands to get information about your launch templates: describe-launch-templates and describe-launch-template-versions.

The describe-launch-templates command enables you to get a list of any of the launch templates that you have created. You can use an option to filter results on a launch template name, create time, tag key, or tag key-value combination. This command returns summary information about any of your launch templates, including the launch template identifier, latest version, and default version.

The following example provides a summary of the specified launch template.

```
aws ec2 describe-launch-templates --launch-template-names my-template-for-auto-scaling
```

The following is an example response.

```
{
    "LaunchTemplates": [
        {
            "LaunchTemplateId": "lt-068f72b729example",
            "LaunchTemplateName": "my-template-for-auto-scaling",
            "CreateTime": "2020-02-28T19:52:27.000Z",
            "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
            "DefaultVersionNumber": 1,
            "LatestVersionNumber": 1
        }
    ]
}
```

If you don't use the `--launch-template-names` option to limit the output to one launch template, information on all of your launch templates is returned.

The following describe-launch-template-versions command provides information describing the versions of the specified launch template.

```
aws ec2 describe-launch-template-versions --launch-template-id lt-068f72b729example
```

The following is an example response.

```
{
    "LaunchTemplateVersions": [
        {
            "VersionDescription": "version1",
            "LaunchTemplateId": "lt-068f72b729example",
            "LaunchTemplateName": "my-template-for-auto-scaling",
            "VersionNumber": 1,
            "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
            "LaunchTemplateData": {
                "TagSpecifications": [
                    {
                        "ResourceType": "instance",
                        "Tags": [
                            {
                                "Key": "purpose",
                                "Value": "webserver"
                            }
                        ]
                    }
                ],
                "ImageId": "ami-04d5cc9b88example",
                "InstanceType": "t2.micro",
                "NetworkInterfaces": [
                    {
                        "DeviceIndex": 0,
                        "DeleteOnTermination": true,
                        "Groups": [
                            "sg-903004f88example"
                        ],
                        "AssociatePublicIpAddress": true
                    }
                ]
            },
            "DefaultVersion": true,
            "CreateTime": "2020-02-28T19:52:27.000Z"
        }
    ]
}
```

Create a launch template version

The following create-launch-template-version command creates a new launch template version based on version 1 of the launch template and specifies a different AMI ID.

```
aws ec2 create-launch-template-version --launch-template-id lt-068f72b729example --
version-description version2 \
  --source-version 1 --launch-template-data "ImageId=ami-c998b6b2example"
```

To set the default version of the launch template, use the modify-launch-template command.

Delete a launch template version

The following delete-launch-template-versions command deletes the specified launch template version.

```
aws ec2 delete-launch-template-versions --launch-template-id lt-068f72b729example --
versions 1
```

Delete a launch template

If you no longer require a launch template, you can delete it using the following delete-launch-template command. Deleting a launch template deletes all of its versions.

```
aws ec2 delete-launch-template --launch-template-id lt-068f72b729example
```

Update an Auto Scaling group to use a launch template

You can use the update-auto-scaling-group command to add a launch template to an existing Auto Scaling group.

Update an Auto Scaling group to use the latest version of a launch template

The following update-auto-scaling-group command updates the specified Auto Scaling group to use the latest version of the specified launch template.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-template LaunchTemplateId=lt-068f72b729example,Version='$Latest'
```

Update an Auto Scaling group to use a specific version of a launch template

The following update-auto-scaling-group command updates the specified Auto Scaling group to use a specific version of the specified launch template.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-template LaunchTemplateName=my-template-for-auto-scaling,Version='2'
```

Use AWS Systems Manager parameters instead of AMI IDs in launch templates

This section shows you how to create a launch template that specifies an AWS Systems Manager parameter that references an Amazon Machine Image (AMI) ID. You can use a parameter stored in your same AWS account, a parameter shared from another AWS account, or a public parameter for a public AMI maintained by AWS.

With Systems Manager parameters, you can update your Auto Scaling groups to use new AMI IDs without needing to create new launch templates or new versions of launch templates each time an AMI ID changes. These IDs can change regularly, such as when an AMI is updated with the latest operating system or software updates.

You can create, update, or delete your own Systems Manager parameters using the Parameter Store, a capability of AWS Systems Manager. You must create a Systems Manager parameter before you can use it in a launch template. To get started, create a parameter with the data type `aws:ec2:image`, and for its value, enter the ID of an AMI. The AMI ID has the form `ami-<identifier>`, for example, `ami-123example456`. The correct AMI ID depends on the instance type and AWS Region that you're launching your Auto Scaling group in.

For more information about creating a valid parameter for an AMI ID, see Creating Systems Manager parameters.

Create a launch template that specifies a parameter for the AMI

To create a launch template that specifies a parameter for the AMI, use one of the following methods:

Console

To create a launch template using an AWS Systems Manager parameter

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Launch Templates**, and then choose **Create launch template**.
3. For **Launch template name**, enter a descriptive name for the launch template.
4. Under **Application and OS Images (Amazon Machine Image)**, choose **Browse more AMIs**.
5. Choose the arrow button to the right of the search bar, and then choose **Specify custom value/Systems Manager parameter**.
6. In the **Specify custom value or Systems Manager parameter** dialog box, do the following:

   a. For **AMI ID or Systems Manager parameter string**, enter the Systems Manager parameter name using one of the following formats:

      To reference a public parameter:

      - `resolve:ssm:public-parameter`

      To reference a parameter stored in the same account:

      - `resolve:ssm:parameter-name`
      - `resolve:ssm:parameter-name:version-number`
      - `resolve:ssm:parameter-name:label`

      To reference a parameter shared from another AWS account:

      - `resolve:ssm:parameter-ARN`
      - `resolve:ssm:parameter-ARN:version-number`
      - `resolve:ssm:parameter-ARN:label`

   b. Choose **Save**.
7. Configure any other launch template settings as needed, and then choose **Create launch template**. For more information, see Create a launch template for an Auto Scaling group.

AWS CLI

To create a launch template that specifies a Systems Manager parameter, you can use one of the following example commands. Replace each *user input placeholder* with your own information.

Example: Create a launch template that specifies an AWS-owned public parameter

Use the following syntax: `resolve:ssm:public-parameter`, where `resolve:ssm` is the standard prefix and *public-parameter* is the path and name of the public parameter.

In this example, the launch template uses an AWS-provided public parameter to launch instances using the latest Amazon Linux 2 AMI in the AWS Region that is configured for your profile.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling
 --version-description version1 \
  --launch-template-data file://config.json
```

Contents of `config.json`:

```
{
    "ImageId":"resolve:ssm:/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-
x86_64-gp2",
    "InstanceType":"t2.micro"
}
```

The following is an example response.

```
{
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-089c023a30example",
        "LaunchTemplateName": "my-template-for-auto-scaling",
        "CreateTime": "2022-12-28T19:52:27.000Z",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
    }
}
```

Example: Create a launch template that specifies a parameter stored in the same account

Use the following syntax: `resolve:ssm:parameter-name`, where `resolve:ssm` is the standard prefix and *`parameter-name`* is the Systems Manager parameter name.

The following example creates a launch template that gets the AMI ID from an existing Systems Manager parameter named *`golden-ami`*.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling \
  --launch-template-data file://config.json
```

Contents of `config.json`:

```
{
    "ImageId":"resolve:ssm:golden-ami",
    "InstanceType":"t2.micro"
}
```

The default version of the parameter, if it is not specified, is the latest version.

The following example references a specific version of the *`golden-ami`* parameter. The example uses version *`3`* of the *`golden-ami`* parameter, but you can use any valid version number.

```
{
    "ImageId":"resolve:ssm:golden-ami:3",
    "InstanceType":"t2.micro"
}
```

The following similar example references the parameter label *`prod`* that maps to a specific version of the *`golden-ami`* parameter.

```
{
    "ImageId":"resolve:ssm:golden-ami:prod",
    "InstanceType":"t2.micro"
}
```

The following is example output.

```
{
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-068f72b724example",
```

```
        "LaunchTemplateName": "my-template-for-auto-scaling",
        "CreateTime": "2022-12-27T17:11:21.000Z",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
    }
}
```

Example: Create a launch template that specifies a parameter shared from another AWS account

Use the following syntax: `resolve:ssm:`*`parameter-ARN`*, where `resolve:ssm` is the standard prefix and *`parameter-ARN`* is the ARN of the Systems Manager parameter.

The following example creates a launch template that gets the AMI ID from an existing Systems Manager parameter with the ARN of *`arn:aws:ssm:us-east-2:123456789012:parameter/MyParameter`*.

```
aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling
 --version-description version1 \
  --launch-template-data file://config.json
```

Contents of `config.json`:

```
{
    "ImageId":"resolve:ssm:arn:aws:ssm:us-east-2:123456789012:parameter/
MyParameter",
    "InstanceType":"t2.micro"
}
```

The default version of the parameter, if it is not specified, is the latest version.

The following example references a specific version of the *`MyParameter`* parameter. The example uses version *`3`* of the *`MyParameter`* parameter, but you can use any valid version number.

```
{
    "ImageId":"resolve:ssm:arn:aws:ssm:us-east-2:123456789012:parameter/
MyParameter:3",
    "InstanceType":"t2.micro"
}
```

The following similar example references the parameter label *prod* that maps to a specific version of the *MyParameter* parameter.

```
{
    "ImageId":"resolve:ssm:arn:aws:ssm:us-east-2:123456789012:parameter/
MyParameter:prod",
    "InstanceType":"t2.micro"
}
```

The following is an example response.

```
{
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-00f93d4588example",
        "LaunchTemplateName": "my-template-for-auto-scaling",
        "CreateTime": "2024-01-08T12:43:21.000Z",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
    }
}
```

To specify a parameter from the Parameter Store in a launch template, you must have the `ssm:GetParameters` permission for the specified parameter. Anyone who uses the launch template also needs the `ssm:GetParameters` permission in order for the parameter value to be validated. For more information, see Restricting access to Systems Manager parameters using IAM policies in the *AWS Systems Manager User Guide*.

Verify a launch template gets the correct AMI ID

Use the describe-launch-template-versions command and include the `--resolve-alias` option to resolve the parameter to the actual AMI ID.

```
aws ec2 describe-launch-template-versions --launch-template-name my-template-for-auto-
scaling \
   --versions $Default --resolve-alias
```

The example returns the AMI ID for `ImageId`. When an instance is launched using this launch template, the AMI ID resolves to `ami-0ac394d6a3example`.

```
{
    "LaunchTemplateVersions": [
        {
            "LaunchTemplateId": "lt-089c023a30example",
            "LaunchTemplateName": "my-template-for-auto-scaling",
            "VersionNumber": 1,
            "CreateTime": "2022-12-28T19:52:27.000Z",
            "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
            "DefaultVersion": true,
            "LaunchTemplateData": {
                "ImageId": "ami-0ac394d6a3example",
                "InstanceType": "t2.micro",
            }
        }
    ]
}
```

Related resources

For more details about specifying a Systems Manager parameter in your launch template, see Use a Systems Manager parameter instead of an AMI ID in the *Amazon EC2 User Guide*.

For more information about working with Systems Manager parameters, see the following reference materials in the Systems Manager documentation.

- To create parameter versions and labels, see Working with parameter versions and Working with parameter labels.
- For information about how to look up the AMI public parameters supported by Amazon EC2, see Calling AMI public parameters.
- For information about sharing parameters with other AWS accounts or through AWS Organizations, see Working with shared parameters.
- For information about monitoring whether your parameters are created successfully, see Native parameter support for Amazon Machine Image IDs.

Limitations

When working with Systems Manager parameters, note the following limitations:

- Amazon EC2 Auto Scaling only supports specifying AMI IDs as parameters.

- Creating or updating mixed instances groups with attribute-based instance type selection using a launch template that specifies a Systems Manager parameter is not supported.
- If your Auto Scaling group uses a launch template that specifies a Systems Manager parameter, you will not be able to start an instance refresh with a desired configuration or using skip matching.
- On each call to create or update your Auto Scaling group, Amazon EC2 Auto Scaling will resolve the Systems Manager parameter in the launch template. If you are using advanced parameters or higher throughput limits, the frequent calls to the Parameter Store (that is, the `GetParameters` operation) can increase your costs for Systems Manager because charges are incurred per Parameter Store API interaction. For more information, see AWS Systems Manager pricing.

Auto Scaling launch configurations

> **Important**
>
> You can not call `CreateLaunchConfiguration` with new Amazon EC2 instance types that are released after **December 31, 2022**. In addition, any new accounts created on or after **June 1, 2023** will not have the option to create new launch configurations through the console. Starting on **October 1, 2024**, new accounts will not be able to create new launch configurations by using the console, API, CLI, and CloudFormation. Migrate to launch templates to ensure that you don't need to create new launch configurations now or in the future. For information about migrating your Auto Scaling groups to launch templates, see Migrate your Auto Scaling groups to launch templates.

A *launch configuration* is an instance configuration template that an Auto Scaling group uses to launch EC2 instances. When you create a launch configuration, you specify information for the instances. Include the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping. If you've launched an EC2 instance before, you specified the same information in order to launch the instance.

You can specify your launch configuration with multiple Auto Scaling groups. However, you can only specify one launch configuration for an Auto Scaling group at a time, and you can't modify a launch configuration after you've created it. To change the launch configuration for an Auto Scaling group, you must create a launch configuration and then update your Auto Scaling group with it.

Contents

- Create a launch configuration
- Change the launch configuration for an Auto Scaling group

Create a launch configuration

> **Important**
>
> You can not call `CreateLaunchConfiguration` with new Amazon EC2 instance types that are released after **December 31, 2022**. In addition, any new accounts created on or

after **June 1, 2023** will not have the option to create new launch configurations through the console. Starting on **October 1, 2024**, new accounts will not be able to create new launch configurations by using the console, API, CLI, and CloudFormation. Migrate to launch templates to ensure that you don't need to create new launch configurations now or in the future. For information about migrating your Auto Scaling groups to launch templates, see Migrate your Auto Scaling groups to launch templates.

This topic describes how to create a launch configuration.

After you create a launch configuration, you cannot modify it. Instead, you must create a new launch configuration.

To associate a new launch configuration with an existing Auto Scaling group, see Change the launch configuration for an Auto Scaling group. To create a new Auto Scaling group, see Create an Auto Scaling group using a launch configuration.

Contents

- Create a launch configuration
- Configure the instance metadata options
- Create a launch configuration using an EC2 instance

Create a launch configuration

To create a launch configuration (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the top navigation bar, select your AWS Region.
3. On the left navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
4. Choose **Launch configurations** near the top of the page. When prompted for confirmation, choose **View launch configurations** to confirm that you want to view the **Launch configurations** page.
5. Choose **Create launch configuration**, and enter a name for your launch configuration.
6. For **Amazon machine image (AMI)** , choose an AMI. To find a specific AMI, you can find a suitable AMI, make note of its ID, and enter the ID as search criteria.

   To get the ID of the Amazon Linux 2 AMI:

a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

b. On the left navigation pane, under **Instances**, choose **Instances**, and then choose **Launch instances**.

c. On the **Quick Start** tab of the **Choose an Amazon Machine Image** page, note the ID of the AMI next to **Amazon Linux 2 AMI (HVM)**.

7. For **Instance type**, select a hardware configuration for your instances.

8. Under **Additional configuration**, pay attention to the following fields:

a. (Optional) For **Purchasing option**, you can choose **Request Spot Instances** to request Spot Instances at the Spot price, capped at the On-Demand price. Optionally, you can specify a maximum price per instance hour for your Spot Instances.

> **Note**
>
> Spot Instances are a cost-effective choice compared to On-Demand Instances, if you can be flexible about when your applications run and if your applications can be interrupted. For more information, see Request Spot Instances for fault-tolerant and flexible applications.

b. (Optional) For **IAM instance profile**, choose a role to associate with the instances. For more information, see IAM role for applications that run on Amazon EC2 instances.

c. (Optional) For **Monitoring**, choose whether to enable the instances to publish metric data at 1-minute intervals to Amazon CloudWatch by enabling detailed monitoring. Additional charges apply. For more information, see Configure monitoring for Auto Scaling instances.

d. (Optional) For **Advanced details**, **User data**, you can specify user data to configure an instance during launch, or to run a configuration script after the instance starts.

e. (Optional) For **Advanced details**, **IP address type**, choose whether to assign a public IP address to the group's instances. If you do not set a value, the default is to use the auto-assign public IP settings of the subnets that your instances are launched into.

9. (Optional) For **Storage (volumes)**, if you don't need additional storage, you can skip this section. Otherwise, to specify volumes to attach to the instances in addition to the volumes specified by the AMI, choose **Add new volume**. Then choose the desired options and associated values for **Devices**, **Snapshot**, **Size**, **Volume type**, **IOPS**, **Throughput**, **Delete on termination**, and **Encrypted**.

10. For **Security groups**, create or select the security group to associate with the group's instances. If you leave the **Create a new security group** option selected, a default SSH rule is configured for Amazon EC2 instances running Linux. A default RDP rule is configured for Amazon EC2 instances running Windows.
11. For **Key pair (login)**, choose an option under **Key pair options**.

    If you've already configured an Amazon EC2 instance key pair, you can choose it here.

    If you don't already have an Amazon EC2 instance key pair, choose **Create a new key pair** and give it a recognizable name. Choose **Download key pair** to download the key pair to your computer.

    > **Important**
    >
    > If you need to connect to your instances, do not choose **Proceed without a key pair**.

12. Select the acknowledgment check box, and then choose **Create launch configuration**.

To create a launch configuration from an existing launch configuration (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the top navigation bar, select your AWS Region.
3. On the left navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
4. Choose **Launch configurations** near the top of the page. When prompted for confirmation, choose **View launch configurations** to confirm that you want to view the **Launch configurations** page.
5. Select the launch configuration and choose **Actions**, **Copy launch configuration**. This sets up a new launch configuration with the same options as the original, but with "Copy" added to the name.
6. On the **Copy Launch Configuration** page, edit the configuration options as needed and choose **Create launch configuration**.

To create a launch configuration using the command line

You can use one of the following commands:

- create-launch-configuration (AWS CLI)

- New-ASLaunchConfiguration (AWS Tools for Windows PowerShell)

Configure the instance metadata options

Amazon EC2 Auto Scaling supports configuring the Instance Metadata Service (IMDS) in launch configurations. This gives you the option of using launch configurations to configure the Amazon EC2 instances in your Auto Scaling groups to require Instance Metadata Service Version 2 (IMDSv2), which is a session-oriented method for requesting instance metadata. For details about IMDSv2's advantages, see this article on the AWS Blog about enhancements to add defense in depth to the EC2 instance metadata service.

You can configure IMDS to support both IMDSv2 and IMDSv1 (the default), or to require the use of IMDSv2. If you are using the AWS CLI or one of the SDKs to configure IMDS, you must use the latest version of the AWS CLI or the SDK to require the use of IMDSv2.

You can configure your launch configuration for the following:

- Require the use of IMDSv2 when requesting instance metadata
- Specify the PUT response hop limit
- Turn off access to instance metadata

You can find more details on configuring the Instance Metadata Service in the following topic: Configuring the instance metadata service in the *Amazon EC2 User Guide*.

Use the following procedure to configure IMDS options in a launch configuration. After you create your launch configuration, you can associate it with your Auto Scaling group. If you associate the launch configuration with an existing Auto Scaling group, the existing launch configuration is disassociated from the Auto Scaling group, and existing instances will require replacement to use the IMDS options that you specified in the new launch configuration. For more information, see Change the launch configuration for an Auto Scaling group.

To configure IMDS in a launch configuration (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the top navigation bar, select your AWS Region.
3. On the left navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.

4. Choose **Launch configurations** near the top of the page. When prompted for confirmation, choose **View launch configurations** to confirm that you want to view the **Launch configurations** page.
5. Choose **Create launch configuration**, and create the launch configuration the usual way. Include the ID of the Amazon Machine Image (AMI), the instance type, and optionally, a key pair, one or more security groups, and any additional EBS volumes or instance store volumes for your instances.
6. To configure instance metadata options for all of the instances associated with this launch configuration, in **Additional configuration**, under **Advanced details**, do the following:

   a. For **Metadata accessible**, choose whether to enable or disable access to the HTTP endpoint of the instance metadata service. By default, the HTTP endpoint is enabled. If you choose to disable the endpoint, access to your instance metadata is turned off. You can specify the condition to require IMDSv2 only when the HTTP endpoint is enabled.

   b. For **Metadata version**, you can choose to require the use of Instance Metadata Service Version 2 (IMDSv2) when requesting instance metadata. If you do not specify a value, the default is to support both IMDSv1 and IMDSv2.

   c. For **Metadata token response hop limit**, you can set the allowable number of network hops for the metadata token. If you do not specify a value, the default is 1.
7. When you have finished, choose **Create launch configuration**.

To require the use of IMDSv2 in a launch configuration using the AWS CLI

Use the following create-launch-configuration command with `--metadata-options` set to `HttpTokens=required`. When you specify a value for `HttpTokens`, you must also set `HttpEndpoint` to enabled. Because the secure token header is set to required for metadata retrieval requests, this opts in the instance to require using IMDSv2 when requesting instance metadata.

```
aws autoscaling create-launch-configuration \
  --launch-configuration-name my-lc-with-imdsv2 \
  --image-id ami-01e24be29428c15b2 \
  --instance-type t2.micro \
      ...
  --metadata-options "HttpEndpoint=enabled,HttpTokens=required"
```

To turn off access to instance metadata

Use the following create-launch-configuration command to turn off access to instance metadata. You can turn access back on later by using the modify-instance-metadata-options command.

```
aws autoscaling create-launch-configuration \
  --launch-configuration-name my-lc-with-imds-disabled \
  --image-id ami-01e24be29428c15b2 \
  --instance-type t2.micro \
      ...
  --metadata-options "HttpEndpoint=disabled"
```

Create a launch configuration using an EC2 instance

You also have the option to create a launch configuration using the attributes from a running EC2 instance.

There are differences between creating a launch configuration from scratch and creating a launch configuration from an existing EC2 instance. When you create a launch configuration from scratch, you specify the image ID, instance type, optional resources (such as storage devices), and optional settings (like monitoring). When you create a launch configuration from a running instance, Amazon EC2 Auto Scaling derives attributes for the launch configuration from the specified instance. Attributes are also derived from the block device mapping for the AMI from which the instance was launched, ignoring any additional block devices that were added after launch.

When you create a launch configuration using a running instance, you can override the following attributes by specifying them as part of the same request: AMI, block devices, key pair, instance profile, instance type, kernel, instance monitoring, placement tenancy, ramdisk, security groups, Spot (max) price, user data, whether the instance has a public IP address, and whether the instance is EBS-optimized.

> **Note**
>
> If the specified instance has properties that are not currently supported by launch configurations, the instances launched by the Auto Scaling group might not be identical to the original EC2 instance.

> **Important**
>
> The AMI used to launch the specified instance must still exist.

Topics

- Create a launch configuration from an EC2 instance (AWS CLI)
- Create a launch configuration from an instance and override the block devices (AWS CLI)
- Create a launch configuration and override the instance type (AWS CLI)

Create a launch configuration from an EC2 instance (AWS CLI)

Use the following create-launch-configuration command to create a launch configuration from an instance using the same attributes as the instance. Any block devices added after launch are ignored.

```
aws autoscaling create-launch-configuration --launch-configuration-name my-lc-from-instance --instance-id i-a8e09d9c
```

You can use the following describe-launch-configurations command to describe the launch configuration and verify that its attributes match those of the instance.

```
aws autoscaling describe-launch-configurations --launch-configuration-names my-lc-from-instance
```

The following is an example response.

```
{
    "LaunchConfigurations": [
        {
            "UserData": null,
            "EbsOptimized": false,
            "LaunchConfigurationARN": "arn",
            "InstanceMonitoring": {
                "Enabled": false
            },
            "ImageId": "ami-05355a6c",
            "CreatedTime": "2014-12-29T16:14:50.382Z",
            "BlockDeviceMappings": [],
            "KeyName": "my-key-pair",
            "SecurityGroups": [
                "sg-8422d1eb"
            ],
            "LaunchConfigurationName": "my-lc-from-instance",
            "KernelId": "null",
```

```
            "RamdiskId": null,
            "InstanceType": "t1.micro",
            "AssociatePublicIpAddress": true
        }
    ]
}
```

Create a launch configuration from an instance and override the block devices (AWS CLI)

By default, Amazon EC2 Auto Scaling uses the attributes from the EC2 instance that you specify to create the launch configuration. However, the block devices come from the AMI used to launch the instance, not the instance. To add block devices to the launch configuration, override the block device mapping for the launch configuration.

Use the following create-launch-configuration command to create a launch configuration using an EC2 instance but with a custom block device mapping.

```
aws autoscaling create-launch-configuration --launch-configuration-name my-lc-from-instance-bdm --instance-id i-a8e09d9c \
  --block-device-mappings "[{\"DeviceName\":\"/dev/sda1\",\"Ebs\":{\"SnapshotId\":\"snap-3decf207\"}},{\"DeviceName\":\"/dev/sdf\",\"Ebs\":{\"SnapshotId\":\"snap-eed6ac86\"}}]"
```

Use the following describe-launch-configurations command to describe the launch configuration and verify that it uses your custom block device mapping.

```
aws autoscaling describe-launch-configurations --launch-configuration-names my-lc-from-instance-bdm
```

The following example response describes the launch configuration.

```
{
    "LaunchConfigurations": [
        {
            "UserData": null,
            "EbsOptimized": false,
            "LaunchConfigurationARN": "arn",
            "InstanceMonitoring": {
                "Enabled": false
            },
```

```
            "ImageId": "ami-c49c0dac",
            "CreatedTime": "2015-01-07T14:51:26.065Z",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/sda1",
                    "Ebs": {
                        "SnapshotId": "snap-3decf207"
                    }
                },
                {
                    "DeviceName": "/dev/sdf",
                    "Ebs": {
                        "SnapshotId": "snap-eed6ac86"
                    }
                }
            ],
            "KeyName": "my-key-pair",
            "SecurityGroups": [
                "sg-8637d3e3"
            ],
            "LaunchConfigurationName": "my-lc-from-instance-bdm",
            "KernelId": null,
            "RamdiskId": null,
            "InstanceType": "t1.micro",
            "AssociatePublicIpAddress": true
        }
    ]
}
```

Create a launch configuration and override the instance type (AWS CLI)

By default, Amazon EC2 Auto Scaling uses the attributes from the EC2 instance you specify to create the launch configuration. Depending on your requirements, you might want to override attributes from the instance and use the values that you need. For example, you can override the instance type.

Use the following create-launch-configuration command to create a launch configuration using an EC2 instance but with a different instance type (for example `t2.medium`) than the instance (for example `t2.micro`).

```
aws autoscaling create-launch-configuration --launch-configuration-name my-lc-from-instance-changetype \
```

```
  --instance-id i-a8e09d9c --instance-type t2.medium
```

Use the following describe-launch-configurations command to describe the launch configuration and verify that the instance type was overridden.

```
aws autoscaling describe-launch-configurations --launch-configuration-names my-lc-from-instance-changetype
```

The following example response describes the launch configuration.

```
{
    "LaunchConfigurations": [
        {
            "UserData": null,
            "EbsOptimized": false,
            "LaunchConfigurationARN": "arn",
            "InstanceMonitoring": {
                "Enabled": false
            },
            "ImageId": "ami-05355a6c",
            "CreatedTime": "2014-12-29T16:14:50.382Z",
            "BlockDeviceMappings": [],
            "KeyName": "my-key-pair",
            "SecurityGroups": [
                "sg-8422d1eb"
            ],
            "LaunchConfigurationName": "my-lc-from-instance-changetype",
            "KernelId": "null",
            "RamdiskId": null,
            "InstanceType": "t2.medium",
            "AssociatePublicIpAddress": true
        }
    ]
}
```

Change the launch configuration for an Auto Scaling group

Important

We provide information about launch configurations for customers who have not yet migrated from launch configurations to launch templates. For information about migrating

your Auto Scaling groups to launch templates, see Migrate your Auto Scaling groups to launch templates.

This topic describes how to associate a different launch configuration with your Auto Scaling group.

After you change the launch configuration, any new instances are launched using the new configuration options, but existing instances are not affected. For more information, see Update Auto Scaling instances.

To change the launch configuration for an Auto Scaling group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the left navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
4. On the **Details** tab, choose **Launch configuration**, **Edit**.
5. For **Launch configuration**, choose the launch configuration.
6. When you have finished, choose **Update**.

To change the launch configuration for an Auto Scaling group using the command line

You can use one of the following commands:

- update-auto-scaling-group (AWS CLI)
- Update-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Auto Scaling groups

> **Note**
>
> If you are new to Auto Scaling groups, work through the steps in the Create your first Auto Scaling group tutorial to get started and see how an Auto Scaling group responds when an instance in the group terminates.

An *Auto Scaling group* contains a collection of EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management. An Auto Scaling group also lets you use Amazon EC2 Auto Scaling features such as health check replacements and scaling policies. Both maintaining the number of instances in an Auto Scaling group and automatic scaling are the core functionality of the Amazon EC2 Auto Scaling service.

The size of an Auto Scaling group depends on the number of instances that you set as the desired capacity. You can adjust its size to meet demand, either manually or by using automatic scaling.

An Auto Scaling group starts by launching enough instances to meet its desired capacity. It maintains this number of instances by performing periodic health checks on the instances in the group. The Auto Scaling group continues to maintain a fixed number of instances even if an instance becomes unhealthy. If an instance becomes unhealthy, the group terminates the unhealthy instance and launches another instance to replace it. For more information, see Health checks for instances in an Auto Scaling group.

You can use scaling policies to increase or decrease the number of instances in your group dynamically to meet changing conditions. When the scaling policy is in effect, the Auto Scaling group adjusts the desired capacity of the group, between the minimum and maximum capacity values that you specify, and launches or terminates the instances as needed. You can also scale on a schedule. For more information, see Choose your scaling method.

When creating an Auto Scaling group, you can choose whether to launch On-Demand Instances, Spot Instances, or both. You can specify multiple purchase options for your Auto Scaling group only when you use a launch template. For more information, see Auto Scaling groups with multiple instance types and purchase options.

Spot Instances provide you with access to unused EC2 capacity at steep discounts relative to On-Demand prices. For more information, see Amazon EC2 Spot Instances. There are key differences between Spot Instances and On-Demand Instances:

- The price for Spot Instances varies based on demand
- Amazon EC2 can terminate an individual Spot Instance as the availability of, or price for, Spot Instances changes

When a Spot Instance is terminated, the Auto Scaling group attempts to launch a replacement instance to maintain the desired capacity for the group.

When instances are launched, if you specified multiple Availability Zones, the desired capacity is distributed across these Availability Zones. If a scaling action occurs, Amazon EC2 Auto Scaling automatically maintains balance across all of the Availability Zones that you specify.

Contents

- Create Auto Scaling groups using launch templates
- Create Auto Scaling groups using launch configurations
- Update an Auto Scaling group
- Tag Auto Scaling groups and instances
- Instance maintenance policies
- Amazon EC2 Auto Scaling lifecycle hooks
- Decrease latency for applications with long boot times using warm pools
- Detach or attach instances from your Auto Scaling group
- Temporarily remove instances from your Auto Scaling group
- Delete your Auto Scaling infrastructure

Create Auto Scaling groups using launch templates

If you have created a launch template, you can create an Auto Scaling group that uses a launch template as a configuration template for its EC2 instances. The launch template specifies information such as the AMI ID, instance type, key pair, security groups, and block device mapping for your instances. For information about creating launch templates, see Create a launch template for an Auto Scaling group.

You must have sufficient permissions to create an Auto Scaling group. You must also have sufficient permissions to create the service-linked role that Amazon EC2 Auto Scaling uses to perform actions on your behalf if it does not yet exist. For examples of IAM policies that an administrator can use as a reference for granting you permissions, see Identity-based policy examples and Control Amazon EC2 launch template usage in Auto Scaling groups.

Contents

- Create an Auto Scaling group using a launch template
- Create an Auto Scaling group using the Amazon EC2 launch wizard
- Auto Scaling groups with multiple instance types and purchase options

Create an Auto Scaling group using a launch template

When you create an Auto Scaling group, you must specify the necessary information to configure the Amazon EC2 instances, the Availability Zones and VPC subnets for the instances, the desired capacity, and the minimum and maximum capacity limits.

To configure Amazon EC2 instances that are launched by your Auto Scaling group, you can specify a launch template or a launch configuration. The following procedure demonstrates how to create an Auto Scaling group using a launch template.

Prerequisites

- You must have created a launch template. For more information, see Create a launch template for an Auto Scaling group.

To create an Auto Scaling group using a launch template (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the same AWS Region that you used when you created the launch template.
3. Choose **Create an Auto Scaling group**.
4. On the **Choose launch template or configuration** page, do the following:

   a. For **Auto Scaling group name**, enter a name for your Auto Scaling group.

b. For **Launch template**, choose an existing launch template.

c. For **Launch template version**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.

d. Verify that your launch template supports all of the options that you are planning to use, and then choose **Next**.

5. On the **Choose instance launch options** page, if you're not using multiple instance types, you can skip the **Instance type requirements** section to use the EC2 instance type that is specified in the launch template.

   To use multiple instance types, see Auto Scaling groups with multiple instance types and purchase options.

6. Under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch template.

7. For **Availability Zones and subnets**, choose one or more subnets in the specified VPC. Use subnets in multiple Availability Zones for high availability. For more information, see Considerations when choosing VPC subnets.

8. If you created a launch template with an instance type specified, then you can continue to the next step to create an Auto Scaling group that uses the instance type in the launch template.

   Alternatively, you can choose the **Override launch template** option if no instance type is specified in your launch template or if you want to use multiple instance types for auto scaling. For more information, see Auto Scaling groups with multiple instance types and purchase options.

9. Choose **Next** to continue to the next step.

   Or, you can accept the rest of the defaults, and choose **Skip to review**.

10. (Optional) On the **Configure advanced options** page, configure the following options, and then choose **Next**:

    a. (Optional) For **Health checks**, **Additional health check types**, select **Turn on Amazon EBS health checks**. For more information, see Monitor Auto Scaling instances with impaired Amazon EBS volumes using health checks.

    b. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.

c. Under **Additional settings**, **Monitoring**, choose whether to enable CloudWatch group metrics collection. These metrics provide measurements that can be indicators of a potential issue, such as number of terminating instances or number of pending instances. For more information, see Monitor CloudWatch metrics for your Auto Scaling groups and instances.

d. For **Enable default instance warmup**, select this option and choose the warmup time for your application. If you are creating an Auto Scaling group that has a scaling policy, the default instance warmup feature improves the Amazon CloudWatch metrics used for dynamic scaling. For more information, see Set the default instance warmup for an Auto Scaling group.

11. (Optional) On the **Configure group size and scaling policies** page, configure the following options, and then choose **Next**:

    a. Under **Group size**, for **Desired capacity**, enter the initial number of instances to launch.

    b. In the **Scaling** section, under **Scaling limits**, if your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed. For more information, see Set scaling limits for your Auto Scaling group.

    c. For **Automatic scaling**, choose whether you want to create a target tracking scaling policy. You can also create this policy after your create your Auto Scaling group.

       If you choose **Target tracking scaling policy**, follow the directions in Create a target tracking scaling policy to create the policy.

    d. For **Instance maintenance policy**, choose whether you want to create an instance maintenance policy. You can also create this policy after your create your Auto Scaling group. Follow the directions in Set an instance maintenance policy to create the policy.

    e. Under **Instance scale-in protection**, choose whether to enable instance scale-in protection. For more information, see Use instance scale-in protection to control instance termination.

12. (Optional) To receive notifications, for **Add notification**, configure the notification, and then choose **Next**. For more information, see Amazon SNS notification options for Amazon EC2 Auto Scaling.

13. (Optional) To add tags, choose **Add tag**, provide a tag key and value for each tag, and then choose **Next**. For more information, see Tag Auto Scaling groups and instances.

14. On the **Review** page, choose **Create Auto Scaling group**.

To create an Auto Scaling group using the command line

You can use one of the following commands:

- create-auto-scaling-group (AWS CLI)
- New-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Create an Auto Scaling group using the Amazon EC2 launch wizard

The following procedure shows how to create an Auto Scaling group by using the **Launch instance** wizard in the Amazon EC2 console. This option automatically populates a launch template with certain configuration details from the **Launch instance** wizard.

> **Note**
>
> The wizard does not populate the Auto Scaling group with the number of instances you specify; it only populates the launch template with the Amazon Machine Image (AMI) ID and instance type. Use the **Create Auto Scaling group** wizard to specify the number of instances to launch.
>
> An AMI provides the information required to configure an instance. You can launch multiple instances from a single AMI when you need multiple instances with the same configuration. We recommend using a custom AMI that already has your application installed on it to avoid having your instances terminated if you reboot an instance belonging to an Auto Scaling group. To use a custom AMI with Amazon EC2 Auto Scaling, you must first create your AMI from a customized instance, and then use the AMI to create a launch template for your Auto Scaling group.

Prerequisites

- You must have created a custom AMI in the same AWS Region where you plan to create the Auto Scaling group. For more information, see Create an AMI in the *Amazon EC2 User Guide*.

Use a custom AMI as a template

In this section, you use the Amazon EC2 launch wizard to automatically populate a launch template with your custom AMI. Alternatively, to set up the launch template from scratch or for more

description of the parameters you can configure for your launch template, see Create your launch template (console).

To use a custom AMI as a template

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation bar at the top of the screen, the current AWS Region is displayed. Select a Region in which to launch your Auto Scaling group.
3. In the navigation pane, choose **Instances**.
4. Choose **Launch instance**, and then do the following:

   a. Under **Name and tags**, leave **Name** blank. The name isn't part of the data that's used to create a launch template.

   b. Under **Application and OS Images (Amazon Machine Image)**, choose **Browse more AMIs** to browse the full AMI catalog.

   c. Choose **My AMIs**, find the AMI that you created, and then choose **Select**.

   d. Under **Instance type**, choose an instance type.

   > **Note**
   >
   > Choose the same instance type that you used when you created the AMI or a more powerful one.

   e. On the right side of the screen, under **Summary**, for **Number of instances**, enter any number. The number that you enter here isn't important. You will specify the number of instances that you want to launch when you create the Auto Scaling group.

      Under the **Number of instances** field, a message displays that says **When launching more than 1 instance, consider EC2 Auto Scaling**.

   f. Choose the **consider EC2 Auto Scaling** hyperlink text.

   g. On the **Launch into Auto Scaling Group** confirmation dialogue, choose **Continue** to go to the **Create launch template** page with the AMI and instance type you selected in the launch instance wizard already populated.

After you choose **Continue**, the **Create launch template** page opens. Follow this procedure to finish creating a launch template.

To create a launch template

1. Under **Launch template name and description**, enter a name and description for the new launch template.
2. (Optional) Under **Key pair (login)**, for **Key pair name**, choose the name of the previously created key pair to use when connecting to instances, for example, using SSH.
3. (Optional) Under **Network settings**, for **Security groups**, choose one or more previously created security groups.
4. (Optional) Under **Configure storage**, update the storage configuration. The default storage configuration is determined by the AMI and the instance type.
5. When you are done configuring the launch template, choose **Create launch template**.
6. On the confirmation page, choose **Create Auto Scaling group**.

Create an Auto Scaling group

> **Note**
>
> The rest of this topic describes the basic procedure for creating an Auto Scaling group. For more description of the parameters you can configure for your Auto Scaling group, see Create an Auto Scaling group using a launch template.

After you choose **Create Auto Scaling group**, the **Create Auto Scaling group** wizard opens. Follow this procedure to create an Auto Scaling group.

To create an Auto Scaling group

1. On the **Choose launch template or configuration** page, enter a name for the Auto Scaling group.
2. The launch template that you created is already selected for you.

   For **Launch template version**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.
3. Choose **Next** to continue to the next step.

4. On the **Choose instance launch options** page, if you're not using multiple instance types, you can skip the **Instance type requirements** section to use the EC2 instance type that is specified in the launch template.

   To use multiple instance types, see Auto Scaling groups with multiple instance types and purchase options.

5. Under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch template.

   > **Tip**
   >
   > If you didn't specify a security group in your launch template, your instances are launched with a default security group from the VPC that you specify. By default, this security group doesn't allow inbound traffic from external networks.

6. For **Availability Zones and subnets**, choose one or more subnets in the specified VPC.
7. Choose **Next** twice to go to the **Configure group size and scaling policies** page.
8. Under **Group size**, define the **Desired capacity** (initial number of instances to launch immediately after the Auto Scaling group is created).
9. In the **Scaling** section, under **Scaling limits**, if your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed. For more information, see Set scaling limits for your Auto Scaling group.
10. Choose **Skip to review**.
11. On the **Review** page, choose **Create Auto Scaling group**.

Next steps

You can check that the Auto Scaling group has been created correctly by viewing the activity history. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched instances. If the instances fail to launch or they launch but then immediately terminate, see the following topics for possible causes and resolutions:

- Troubleshoot Amazon EC2 Auto Scaling: EC2 instance launch failures
- Troubleshoot Amazon EC2 Auto Scaling: AMI issues
- Troubleshoot unhealthy instances in Amazon EC2 Auto Scaling

You can now attach a load balancer in the same Region as your Auto Scaling group, if desired. For more information, see Use Elastic Load Balancing; to distribute incoming application traffic in your Auto Scaling group .

Auto Scaling groups with multiple instance types and purchase options

You can launch and automatically scale a fleet of On-Demand Instances and Spot Instances within a single Auto Scaling group. In addition to receiving discounts for using Spot Instances, you can use Reserved Instances or a Savings Plan to receive discounts on the regular On-Demand Instance pricing. These factors help you optimize your cost savings for EC2 instances and get the desired scale and performance for your application.

Spot Instances are spare capacity available at steep discounts compared to the EC2 On-Demand price. Spot Instances are a cost-effective choice if you can be flexible about when your applications run and if your applications can be interrupted. They can be used for various fault-tolerant and flexible applications. Examples include stateless web servers, API endpoints, big data and analytics applications, containerized workloads, CI/CD pipelines, high performance and high throughput computing (HPC/HTC), rendering workloads, and other flexible workloads.

For more information, see Instance purchasing options in the *Amazon EC2 User Guide*.

Topics

- Setup overview for creating a mixed instances group
- Allocation strategies for multiple instance types
- Create mixed instances group using attribute-based instance type selection
- Create a mixed instances group by manually choosing instance types
- Configure an Auto Scaling group to use instance weights
- Use a different launch template for an instance type

Setup overview for creating a mixed instances group

This topic provides an overview and best practices for creating an Auto Scaling mixed instances group.

Contents

- Overview

- Instance type flexibility
- Availability Zone flexibility
- Spot max price
- Proactive capacity rebalancing
- Scaling behavior
- Regional availability of instance types
- Related resources
- Limitations

Overview

To create a mixed instances group, you have two options:

- Attribute-based instance type selection – Define your compute requirements to choose your instance types automatically based on their specific instance attributes.
- Manual instance type selection – Manually choose the instance types that suit your workload.

Manual selection

The following steps describe how to create a mixed instances group by manually choosing instance types:

1. Choose a launch template that has the parameters to launch an EC2 instance. Parameters in launch templates are optional, but Amazon EC2 Auto Scaling can't launch an instance if the Amazon Machine Image (AMI) ID is missing from the launch template.
2. Choose the option to override the launch template.
3. Manually choose the instance types that suit your workload.
4. Specify the percentages of On-Demand Instances and Spot Instances to launch.
5. Choose allocation strategies that determine how Amazon EC2 Auto Scaling fulfills your On-Demand and Spot capacities from the possible instance types.
6. Choose the Availability Zones and VPC subnets to launch your instances in.
7. Specify the initial size of the group (the desired capacity) and the minimum and maximum size of the group.

Overrides are necessary to override the instance type declared in the launch template and use multiple instances types that are embedded in the Auto Scaling group's own resource definition. For more information about the instance types that are available, see Instance types in the *Amazon EC2 User Guide*.

You can also configure the following optional parameters for each instance type:

- `LaunchTemplateSpecification` – You can assign a different launch template to an instance type as needed. This option is currently not available from the console. For more information, see Use a different launch template for an instance type.
- `WeightedCapacity` – You decide how much the instance counts toward the desired capacity relative to the rest of the instances in your group. If you specify a `WeightedCapacity` value for one instance type, you must specify a `WeightedCapacity` value for all of them. By default, each instance counts as one toward your desired capacity. For more information, see Configure an Auto Scaling group to use instance weights.

Attribute-based selection

To let Amazon EC2 Auto Scaling choose your instance types automatically based on their specific instance attributes, use the following steps to create a mixed instances group by specifying your compute requirements:

1. Choose a launch template that has the parameters to launch an EC2 instance. Parameters in launch templates are optional, but Amazon EC2 Auto Scaling can't launch an instance if the Amazon Machine Image (AMI) ID is missing from the launch template.
2. Choose the option to override the launch template.
3. Specify instance attributes that match your compute requirements, such as vCPUs and memory requirements.
4. Specify the percentages of On-Demand Instances and Spot Instances to launch.
5. Choose allocation strategies that determine how Amazon EC2 Auto Scaling fulfills your On-Demand and Spot capacities from the possible instance types.
6. Choose the Availability Zones and VPC subnets to launch your instances in.
7. Specify the initial size of the group (the desired capacity) and the minimum and maximum size of the group.

Overrides are necessary to override the instance type declared in the launch template and use a set of instance attributes that describe your compute requirements. For supported attributes, see InstanceRequirements in the *Amazon EC2 Auto Scaling API Reference*. Alternatively, you can use a launch template that already has your instance attributes definition.

You can also configure the `LaunchTemplateSpecification` parameter within the overrides structure to assign a different launch template to a set of instance requirements as needed. This option is currently not available from the console. For more information, see LaunchTemplateOverrides in the *Amazon EC2 Auto Scaling API Reference*.

By default, you set the number of instances as the desired capacity of your Auto Scaling group.

Alternatively, you can set the value for desired capacity to the number of vCPUs or the amount of memory. To do so, use the `DesiredCapacityType` property in the `CreateAutoScalingGroup` API operation or the **Desired capacity type** dropdown field in the AWS Management Console. This is a useful alternative to instance weights.

Instance type flexibility

To enhance availability, deploy your application across multiple instance types. It's a best practice to use multiple instance types to satisfy capacity requirements. This way, Amazon EC2 Auto Scaling can launch another instance type if there is insufficient instance capacity in your chosen Availability Zones.

If there is insufficient instance capacity with Spot Instances, Amazon EC2 Auto Scaling keeps trying to launch from other Spot Instance pools. (The pools it uses are determined by your choice of instance types and allocation strategy.) Amazon EC2 Auto Scaling helps you leverage the cost savings of Spot Instances by launching them instead of On-Demand Instances.

We recommend being flexible across at least 10 instance types for each workload. When choosing your instance types, don't limit yourself to the most popular new instance types. Choosing earlier generation instance types tends to result in fewer Spot interruptions because they are less in demand from On-Demand customers.

Availability Zone flexibility

We strongly recommend that you span your Auto Scaling group across multiple Availability Zones. With multiple Availability Zones, you can design applications that automatically fail over between zones for greater resiliency.

As an added benefit, you can access a deeper Amazon EC2 capacity pool when compared to groups in a single Availability Zone. Because capacity fluctuates independently for each instance type in each Availability Zone, you can often get more compute capacity with flexibility for both the instance type and the Availability Zone.

For more information about using multiple Availability Zones, see Example: Distribute instances across Availability Zones.

Spot max price

When you create your Auto Scaling group using the AWS CLI or an SDK, you can specify the `SpotMaxPrice` parameter. The `SpotMaxPrice` parameter determines the maximum price that you're willing to pay for a Spot Instance hour.

When you specify the `WeightedCapacity` parameter in your overrides (or `"DesiredCapacityType": "vcpu"` or `"DesiredCapacityType": "memory-mib"` at the group level), the maximum price represents the maximum unit price, not the maximum price for a whole instance.

We strongly recommend that you do not specify a maximum price. Your application might not run if you do not receive any Spot Instances, such as when your maximum price is too low. If you don't specify a maximum price, the default maximum price is the On-Demand price. You pay only the Spot price for the Spot Instances that you launch. You still receive the steep discounts provided by Spot Instances. These discounts are possible because of the stable Spot pricing that's available with the Spot pricing model. For more information, see Pricing and savings in the *Amazon EC2 User Guide*.

Proactive capacity rebalancing

If your use case allows, we recommend Capacity Rebalancing. Capacity Rebalancing helps you maintain workload availability by proactively augmenting your fleet with a new Spot Instance before a running Spot Instance receives the two-minute Spot Instance interruption notice.

When Capacity Rebalancing is enabled, Amazon EC2 Auto Scaling attempts to proactively replace Spot Instances that have received a rebalance recommendation. This provides an opportunity to rebalance your workload to new Spot Instances that are not at elevated risk of interruption.

For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

Scaling behavior

When you create a mixed instances group, it uses On-Demand Instances by default. To use Spot Instances, you must modify the percentage of the group to be launched as On-Demand Instances. You can specify any number from 0 to 100 for the On-Demand percentage.

Optionally, you can also designate a base number of On-Demand Instances to start with. If you do so, Amazon EC2 Auto Scaling waits to launch Spot Instances until after it launches the base capacity of On-Demand Instances when the group scales out. Anything beyond the base capacity uses the On-Demand percentage to determine how many On-Demand Instances and Spot Instances to launch.

Amazon EC2 Auto Scaling converts the percentage to the equivalent number of instances. If the result creates a fractional number, it rounds up to the next integer in favor of On-Demand Instances.

The following table demonstrates the behavior of the Auto Scaling group as it increases and decreases in size.

Example: Scaling behavior

Purchase options	**Group size and number of running instances across purchase options**			
	10	**20**	**30**	**40**
Example 1: base of 10, 50/50% On-Demand/ Spot				
On-Demand Instances (base amount)	10	10	10	10
On-Demand Instances	0	5	10	15
Spot Instances	0	5	10	15

Purchase options	Group size and number of running instances across purchase options			
Example 2: base of 0, 0/100% On-Demand/ Spot				
On-Demand Instances (base amount)	0	0	0	0
On-Demand Instances	0	0	0	0
Spot Instances	10	20	30	40
Example 3: base of 0, 60/40% On-Demand/ Spot				
On-Demand Instances (base amount)	0	0	0	0
On-Demand Instances	6	12	18	24
Spot Instances	4	8	12	16
Example 4: base of 0, 100/0% On-Demand/ Spot				
On-Demand Instances (base amount)	0	0	0	0

Purchase options	Group size and number of running instances across purchase options			
On-Demand Instances	10	20	30	40
Spot Instances	0	0	0	0
Example 5: base of 12, 0/100% On-Demand/ Spot				
On-Demand Instances (base amount)	10	12	12	12
On-Demand Instances	0	0	0	0
Spot Instances	0	8	18	28

When the size of the group *increases*, Amazon EC2 Auto Scaling attempts to balance your capacity evenly across your specified Availability Zones. Then, it launches instance types according to the specified allocation strategy.

When the size of the group *decreases*, Amazon EC2 Auto Scaling first identifies which of the two types (Spot or On-Demand) should be terminated. Then, it tries to terminate instances in a balanced way across your specified Availability Zones. It also favors terminating instances in a way that aligns closer to your allocation strategies. For information about termination policies, see Configure termination policies for Amazon EC2 Auto Scaling.

Regional availability of instance types

The availability of EC2 instance types varies depending on your AWS Region. For example, the newest generation instance types might not yet be available in a given Region. Due to the variances in instance availability across Regions, you might encounter issues when making programmatic requests if multiple instance types in your overrides are not available in your Region. Using multiple instance types that are not available in your Region might cause the request to fail

entirely. To solve the issue, retry the request with different instance types, making sure that each instance type is available in the Region. To search for instance types offered by location, use the describe-instance-type-offerings command. For more information, see Finding an Amazon EC2 instance type in the *Amazon EC2 User Guide*.

Related resources

For more best practices for Spot Instances, see Best practices for EC2 Spot in the *Amazon EC2 User Guide*.

Limitations

After you add overrides to an Auto Scaling group using a mixed instances policy, you can update the overrides with the `UpdateAutoScalingGroup` API call but not delete them. To completely remove the overrides, you must first switch the Auto Scaling group to use a launch template or launch configuration instead of a mixed instances policy. Then, you can add a mixed instances policy again without any overrides.

Allocation strategies for multiple instance types

When you use multiple instance types, you manage how Amazon EC2 Auto Scaling fulfills your On-Demand and Spot capacities from the possible instance types. To do this, you specify allocation strategies.

To review the best practices for a mixed instances group, see Setup overview for creating a mixed instances group.

Contents

- Spot Instances
- On-Demand Instances
- How the allocation strategies work with weights

Spot Instances

Amazon EC2 Auto Scaling provides the following allocation strategies for Spot Instances:

`price-capacity-optimized` (recommended)

The price and capacity optimized allocation strategy looks at both price and capacity to select the Spot Instance pools that are the least likely to be interrupted and have the lowest possible price.

We recommend this strategy when you're getting started. For more information, see Introducing the price-capacity-optimized allocation strategy for EC2 Spot Instances in the AWS blog.

`capacity-optimized`

Amazon EC2 Auto Scaling requests your Spot Instance from the pool with optimal capacity for the number of instances that are launching.

With Spot Instances, the pricing changes slowly over time based on long-term trends in supply and demand. However, capacity fluctuates in real time. The `capacity-optimized` strategy automatically launches Spot Instances into the most available pools by looking at real-time capacity data and predicting which are the most available. This helps to minimize possible disruptions for workloads that might have a higher cost of interruption associated with restarting work and checkpointing. To give certain instance types a higher chance of launching first, use `capacity-optimized-prioritized`.

`capacity-optimized-prioritized`

You set the order of instance types for the launch template overrides from highest to lowest priority (from first to last in the list). Amazon EC2 Auto Scaling honors the instance type priorities on a best-effort basis but optimizes for capacity first. This is a good option for workloads where the possibility of disruption must be minimized, but the preference for certain instance types matters, too. If the On-Demand allocation strategy is set to `prioritized`, the same priority is applied when fulfilling On-Demand capacity.

`lowest-price`

Amazon EC2 Auto Scaling requests your Spot Instances using the lowest priced pools within an Availability Zone, across the N number of Spot pools that you specify for the **Lowest priced pools** setting. For example, if you specify four instance types and four Availability Zones, your Auto Scaling group can access up to 16 Spot pools. (Four in each Availability Zone.) If you specify two Spot pools (N=2) for the allocation strategy, your Auto Scaling group can draw on the two lowest priced pools per Availability Zone to fulfill your Spot capacity.

Because this strategy only considers instance price and not capacity availability, it might lead to high interruption rates.

Amazon EC2 Auto Scaling makes an effort to draw Spot Instances from the N number of pools that you specify. However, if a pool runs out of Spot capacity before fulfilling your desired capacity, Amazon EC2 Auto Scaling continues to fulfill your request by drawing from the next lowest priced pool. To meet your desired capacity, you might receive Spot Instances from more pools than your specified N number. Likewise, if most of the pools have no Spot capacity, you might receive your full desired capacity from fewer pools than your specified N number.

> **Note**
>
> If you configure your Spot Instances to launch with AMD SEV-SNP turned on, you are charged an additional hourly usage fee that is equivalent to 10% of the On-Demand hourly rate of the selected instance type. If the allocation strategy uses price as an input, Amazon EC2 Auto Scaling does not include this additional fee; only the Spot price is used.

On-Demand Instances

Amazon EC2 Auto Scaling provides the following allocation strategies that can be used for On-Demand Instances:

`lowest-price`

Amazon EC2 Auto Scaling automatically deploys the lowest priced instance type in each Availability Zone based on the current On-Demand price.

To meet your desired capacity, you might receive On-Demand Instances of more than one instance type in each Availability Zone. This depends on how much capacity you request.

`prioritized`

When fulfilling On-Demand capacity, Amazon EC2 Auto Scaling determines which instance type to use first based on the order of instance types in the list of launch template overrides. For example, let's say that you specify three launch template overrides in the following order: `c5.large`, `c4.large`, and `c3.large`. When your On-Demand Instances launch, the Auto Scaling group fulfills On-Demand capacity in the following order: `c5.large`, `c4.large`, and then `c3.large`.

Consider the following when managing the priority order of your On-Demand Instances:

- You can pay for usage upfront to get significant discounts for On-Demand Instances by using Savings Plans or Reserved Instances. For more information, see the Amazon EC2 pricing page.
- With Reserved Instances, your discounted rate of the regular On-Demand Instance pricing applies if Amazon EC2 Auto Scaling launches matching instance types. Therefore, if you have unused Reserved Instances for `c4.large`, you can set the instance type priority to give the highest priority for your Reserved Instances to a `c4.large` instance type. When a `c4.large` instance launches, you receive the Reserved Instance pricing.
- With Savings Plans, your discounted rate of the regular On-Demand Instance pricing applies when using Amazon EC2 Instance Savings Plans or Compute Savings Plans. With Savings Plans, you have more flexibility when prioritizing your instance types. As long as you use instance types that are covered by your Savings Plan, you can set them in any priority order. You can also occasionally change the entire order of your instance types, while still receiving the Savings Plan discounted rate. For more information about Savings Plans, see the Savings Plans User Guide.

How the allocation strategies work with weights

When you specify the `WeightedCapacity` parameter in your overrides (or `"DesiredCapacityType": "vcpu"` or `"DesiredCapacityType": "memory-mib"` at the group level), the allocation strategies work exactly like they do for other Auto Scaling groups.

The only difference is that when you choose the `lowest-price` or `price-capacity-optimized` strategy, your instances come from the instance pools with the lowest price per *unit* in each Availability Zone. For more information, see Configure an Auto Scaling group to use instance weights.

For example, imagine you have an Auto Scaling group with several instance types that have varying amounts of vCPUs. You use `lowest-price` for your Spot and On-Demand allocation strategies. If you choose to assign weights based on the vCPU count of each instance type, Amazon EC2 Auto Scaling launches whichever instance types have the lowest price per your assigned weight values (for example, per vCPU) at the time of fulfillment. If it's a Spot Instance, then this means the lowest Spot price per vCPU. If it's an On-Demand Instance, then this means the lowest On-Demand price per vCPU.

Create mixed instances group using attribute-based instance type selection

Instead of manually choosing instance types for your mixed instances group, you can specify a set of instance attributes that describe your compute requirements. As Amazon EC2 Auto Scaling launches instances, any instance types used by the Auto Scaling group must match your required instance attributes. This is known as *attribute-based instance type selection*.

This approach is ideal for workloads and frameworks that can be flexible about which instance types they use, such as containers, big data, and CI/CD.

The following are benefits of attribute-based instance type selection:

- **Optimal flexibility for Spot Instances** – Amazon EC2 Auto Scaling can select from a wide range of instance types for launching Spot Instances. This meets the Spot best practice of being flexible about instance types, which gives the Amazon EC2 Spot service a better chance of finding and allocating your required amount of compute capacity.
- **Easily use the right instance types** – With so many instance types available, finding the right instance types for your workload can be time consuming. When you specify instance attributes, the instance types will automatically have the required attributes for your workload.
- **Automatic use of new instance types** – Your Auto Scaling groups can use newer generation instance types as they're released. Newer generation instance types are automatically used when they match your requirements and align with the allocation strategies you choose for your Auto Scaling group.

Topics

- How attribute-based instance type selection works
- Price protection
- Prerequisites
- Create a mixed instances group with attribute-based instance type selection (console)
- Create a mixed instances group with attribute-based instance type selection (AWS CLI)
- Example configuration
- Preview your instance types
- Related resources

How attribute-based instance type selection works

With attribute-based instance type selection, instead of providing a list of specific instance types, you provide a list of instance attributes that your instances require, such as:

- **vCPU count** – The minimum and maximum number of vCPUs per instance.
- **Memory** – The minimum and maximum GiBs of memory per instance.
- **Local storage** – Whether to use EBS or instance store volumes for local storage.
- **Burstable performance** – Whether to use the T instance family, including T4g, T3a, T3, and T2 types.

There are many options available for defining your instance requirements. For a description of each option and the default values, see InstanceRequirements in the *Amazon EC2 Auto Scaling API Reference*.

When your Auto Scaling group needs to launch an instance, it will search for instance types that match your specified attributes and are available in that Availability Zone. The allocation strategy then determines which of the matching instance types to launch. By default, attribute-based instance type selection has a price protection feature enabled to prevent your Auto Scaling group from launching instance types that exceed your budget thresholds.

By default, you use the number of instances as the unit of measurement when setting the desired capacity of your Auto Scaling group, meaning each instance counts as one unit.

Alternatively, you can set the value for desired capacity to the number of vCPUs or the amount of memory. To do so, use the **Desired capacity type** dropdown field in the AWS Management Console or the `DesiredCapacityType` property in the `CreateAutoScalingGroup` or `UpdateAutoScalingGroup` API operation. Amazon EC2 Auto Scaling then launches the number of instances required to meet the desired vCPU or memory capacity. For example, if you use vCPUs as the desired capacity type and use instances with 2 vCPUs each, a desired capacity of 10 vCPUs would launch 5 instances. This is a useful alternative to instance weights.

Price protection

With price protection, you can specify the maximum price you are willing to pay for EC2 instances launched by your Auto Scaling group. Price protection is a feature that prevents your Auto Scaling group from using instance types that you would consider too expensive even if they happen to fit the attributes that you specified.

Price protection is enabled by default and has separate price thresholds for On-Demand Instances and Spot Instances. When Amazon EC2 Auto Scaling needs to launch new instances, any instance types priced above the relevant threshold are not launched.

Topics

- On-Demand price protection
- Spot price protection
- Customize price protection

On-Demand price protection

For On-Demand Instances, you define the maximum On-Demand price you're willing to pay as a percentage higher than an identified On-Demand price. The identified On-Demand price is the price of the lowest priced current generation C, M, or R instance type with your specified attributes.

If an On-Demand price protection value is not explicitly defined, a default maximum On-Demand price of 20 percent higher than the identified On-Demand price will be used.

Spot price protection

By default, Amazon EC2 Auto Scaling will automatically apply optimal Spot Instance price protection to consistently select from a wide range of instance types. You can also manually set the price protection yourself. However, letting Amazon EC2 Auto Scaling do it for you can improve the likelihood that your Spot capacity is fulfilled.

You can manually specify the price protection using one of the following options. If you manually set the price protection, we recommend using the first option.

- **A percentage of an identified *On-Demand price*** – The identified On-Demand price is the price of the lowest priced current generation C, M, or R instance type with your specified attributes.
- **A percentage higher than an identified *Spot price*** – The identified Spot price is the price of the lowest priced current generation C, M, or R instance type with your specified attributes. We do not recommend using this option because Spot prices can fluctuate, and therefore your price protection threshold might also fluctuate.

Customize price protection

You can customize the price protection thresholds in the Amazon EC2 Auto Scaling console or using the AWS CLI or SDKs.

- In the console, use the **On-Demand price protection** and **Spot price protection** settings in **Additional instance attributes**.
- In the InstanceRequirements structure, to specify the On-Demand Instance price protection threshold, use the `OnDemandMaxPricePercentageOverLowestPrice` property. To specify the Spot Instance price protection threshold, use either the `MaxSpotPriceAsPercentageOfOptimalOnDemandPrice` or the `SpotMaxPricePercentageOverLowestPrice` property.

If you set **Desired capacity type** (`DesiredCapacityType`) to **vCPUs** or **Memory GiB**, the price protection applies based on the per vCPU or per memory price instead of the per instance price.

You can also turn off price protection. To indicate no price protection threshold, specify a high percentage value, such as 999999.

> **Note**
>
> If no current generation C, M, or R instance types match your specified attributes, price protection is still applicable. When no match is found, the identified price is from the lowest priced current generation instance types, or failing that, the lowest priced previous generation instance types, that match your attributes.

Prerequisites

- Create a launch template. For more information, see Create a launch template for an Auto Scaling group.
- Verify that the launch template doesn't already request Spot Instances.

Create a mixed instances group with attribute-based instance type selection (console)

Use the following procedure to create a mixed instances group by using attribute-based instance type selection. To help you move through the steps efficiently, some optional sections are skipped.

For most general purpose workloads, it's enough to specify the number of vCPUs and memory that you need. For advanced use cases, you can specify attributes like storage type, network interfaces, CPU manufacturer, and accelerator type.

To review the best practices for a mixed instances group, see Setup overview for creating a mixed instances group.

To create a mixed instances group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the same AWS Region that you used when you created the launch template.
3. Choose **Create an Auto Scaling group**.
4. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter a name for your Auto Scaling group.
5. To choose your launch template, do the following:
   a. For **Launch template**, choose an existing launch template.
   b. For **Launch template version**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.
   c. Verify that your launch template supports all of the options that you are planning to use, and then choose **Next**.
6. On the **Choose instance launch options** page, do the following:
   a. For **Instance type requirements**, choose **Override launch template**.

      > **Note**
      > If you chose a launch template that already contains a set of instance attributes, such as vCPUs and memory, then the instance attributes are displayed. These attributes are added to the Auto Scaling group properties, where you can update them from the Amazon EC2 Auto Scaling console at any time.

   b. Under **Specify instance attributes**, start by entering your vCPUs and memory requirements.

- For **vCPUs**, enter the desired minimum and maximum number of vCPUs. To specify no limit, select **No minimum**, **No maximum**, or both.
- For **Memory (GiB)**, enter the desired minimum and maximum amount of memory. To specify no limit, select **No minimum**, **No maximum**, or both.

c. (Optional) For **Additional instance attributes**, you can optionally specify one or more attributes to express your compute requirements in more detail. Each additional attribute adds further constraints to your request.

d. Expand **Preview matching instance types** to view the instance types that have your specified attributes.

e. Under **Instance purchase options**, for **Instances distribution**, specify the percentages of the group to launch as On-Demand Instances and as Spot Instances. If your application is stateless, fault tolerant, and can handle an instance being interrupted, you can specify a higher percentage of Spot Instances.

f. (Optional) When you specify a percentage for Spot Instances, select **Include On-Demand base capacity** and then specify the minimum amount of the Auto Scaling group's initial capacity that must be fulfilled by On-Demand Instances. Anything beyond the base capacity uses the **Instances distribution** settings to determine how many On-Demand Instances and Spot Instances to launch.

g. Under **Allocation strategies**, **Lowest price** is automatically selected for the **On-Demand allocation strategy** and cannot be changed.

h. For **Spot allocation strategy**, choose an allocation strategy. **Price capacity optimized** is selected by default. **Lowest price** is hidden by default and only appears when you choose **Show all strategies**. If you choose **Lowest price**, enter the number of lowest priced pools to diversify across for **Lowest priced pools**.

i. For **Capacity Rebalancing**, choose whether to enable or disable Capacity Rebalancing. Use Capacity Rebalancing to automatically respond when your Spot Instances approach termination from a Spot interruption. For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

j. Under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch template.

k. For **Availability Zones and subnets**, choose one or more subnets in the specified VPC. Use subnets in multiple Availability Zones for high availability. For more information, see Considerations when choosing VPC subnets.

l. Choose **Next**, **Next**.

7. For the **Configure group size and scaling policies** step, do the following:

    a. To measure your desired capacity in units other than instances, choose the appropriate option for **Group size**, **Desired capacity type**. **Units**, **vCPUs**, and **Memory GiB** are supported. By default, Amazon EC2 Auto Scaling specifies **Units**, which translates into number of instances.

    b. For **Desired capacity**, the initial size of your Auto Scaling group.

    c. In the **Scaling** section, under **Scaling limits**, if your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed. For more information, see Set scaling limits for your Auto Scaling group.

8. Choose **Skip to review**.

9. On the **Review** page, choose **Create Auto Scaling group**.

Create a mixed instances group with attribute-based instance type selection (AWS CLI)

To create a mixed instances group using the command line

Use one of the following commands:

- create-auto-scaling-group (AWS CLI)
- New-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Example configuration

To create an Auto Scaling group with attribute-based instance type selection by using the AWS CLI, use the following create-auto-scaling-group command.

The following instance attributes are specified:

- `VCpuCount` – The instance types must have a minimum of four vCPUs and a maximum of eight vCPUs.
- `MemoryMiB` – The instance types must have a minimum of 16,384 MiB of memory.
- `CpuManufacturers` – The instance types must have an Intel manufactured CPU.

JSON

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The following is an example `config.json` file.

```
{
    "AutoScalingGroupName": "my-asg",
    "DesiredCapacityType": "units",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Default"
            },
            "Overrides": [{
                "InstanceRequirements": {
                    "VCpuCount": {"Min": 4, "Max": 8},
                    "MemoryMiB": {"Min": 16384},
                    "CpuManufacturers": ["intel"]
                }
            }]
        },
        "InstancesDistribution": {
            "OnDemandPercentageAboveBaseCapacity": 50,
            "SpotAllocationStrategy": "price-capacity-optimized"
        }
    },
    "MinSize": 0,
    "MaxSize": 100,
    "DesiredCapacity": 4,
    "DesiredCapacityType": "units",
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
}
```

To set the value for desired capacity as the number of vCPUs or the amount of memory, specify `"DesiredCapacityType": "vcpu"` or `"DesiredCapacityType": "memory-mib"` in the file. The default desired capacity type is `units`, which sets the value for desired capacity as the number of instances.

YAML

Alternatively, you can use the following create-auto-scaling-group command to create the Auto Scaling group. This references a YAML file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-yaml file://~/config.yaml
```

The following is an example `config.yaml` file.

```
---
AutoScalingGroupName: my-asg
DesiredCapacityType: units
MixedInstancesPolicy:
  LaunchTemplate:
    LaunchTemplateSpecification:
      LaunchTemplateName: my-launch-template
      Version: $Default
    Overrides:
    - InstanceRequirements:
        VCpuCount:
          Min: 2
          Max: 4
        MemoryMiB:
          Min: 2048
        CpuManufacturers:
        - intel
  InstancesDistribution:
    OnDemandPercentageAboveBaseCapacity: 50
    SpotAllocationStrategy: price-capacity-optimized
MinSize: 0
MaxSize: 100
DesiredCapacity: 4
DesiredCapacityType: units
VPCZoneIdentifier: subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782
```

To set the value for desired capacity as the number of vCPUs or the amount of memory, specify `DesiredCapacityType: vcpu` or `DesiredCapacityType: memory-mib` in the file. The default desired capacity type is `units`, which sets the value for desired capacity as the number of instances.

Preview your instance types

You can preview the instance types that match your compute requirements without launching them and adjust your requirements if necessary. When creating your Auto Scaling group in the Amazon EC2 Auto Scaling console, a preview of the instance types appears in the **Preview matching instance types** section on the **Choose instance launch options** page.

Alternatively, you can preview the instance types by making an Amazon EC2 GetInstanceTypesFromInstanceRequirements API call using the AWS CLI or an SDK. Pass the `InstanceRequirements` parameters in the request in the exact format that you would use to create or update an Auto Scaling group. For more information, see Preview instance types with specified attributes in the *Amazon EC2 User Guide*.

Related resources

To learn more about attribute-based instance type selection, see Attribute-Based Instance Type Selection for EC2 Auto Scaling and EC2 Fleet on the AWS Blog.

You can declare attribute-based instance type selection when you create an Auto Scaling group using AWS CloudFormation. For more information, see the example snippet in the Auto scaling template snippets section of the *AWS CloudFormation User Guide*.

Create a mixed instances group by manually choosing instance types

This topic shows you how to launch multiple instance types in a single Auto Scaling group by manually choosing your instance types.

If you prefer to use instance attributes as criteria for selecting instance types, see Create mixed instances group using attribute-based instance type selection.

Contents

- Prerequisites
- Create a mixed instances group (console)
- Create a mixed instances group (AWS CLI)
- Example configurations

Prerequisites

- Create a launch template. For more information, see Create a launch template for an Auto Scaling group.

- Verify that the launch template doesn't already request Spot Instances.

Create a mixed instances group (console)

Use the following procedure to create a mixed instances group by manually choosing which instance types your group can launch. To help you move through the steps efficiently, some optional sections are skipped.

To review the best practices for a mixed instances group, see Setup overview for creating a mixed instances group.

To create a mixed instances group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the same AWS Region that you used when you created the launch template.
3. Choose **Create an Auto Scaling group**.
4. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter a name for your Auto Scaling group.
5. To choose your launch template, do the following:
   a. For **Launch template**, choose an existing launch template.
   b. For **Launch template version**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.
   c. Verify that your launch template supports all of the options that you plan to use, and then choose **Next**.
6. On the **Choose instance launch options** page, do the following:
   a. For **Instance type requirements**, choose **Override launch template**, and then choose **Manually add instance types**.
   b. Choose your instance types. You can use our recommendations as a starting point. **Family and generation flexible** is selected by default.
      - To change the order of the instance types, use the arrows. If you choose an allocation strategy that supports prioritization, the instance type order sets their launch priority.
      - To remove an instance type, choose **X**.

- (Optional) For the boxes in the **Weight** column, assign each instance type a relative weight. To do so, enter the number of units that an instance of that type counts toward the desired capacity of the group. Doing so might be useful if the instance types offer different vCPU, memory, storage, or network bandwidth capabilities. For more information, see Configure an Auto Scaling group to use instance weights.

  Note that if you choose to use **Size flexible** recommendations, then all instance types that are part of this section automatically have a weight value. If you don't want to specify any weights, clear the boxes in the **Weight** column for all instance types.

c. Under **Instance purchase options**, for **Instances distribution**, specify the percentages of the group to be launched as On-Demand Instances and Spot Instances respectively. If your application is stateless, fault tolerant, and can handle an instance being interrupted, you can specify a higher percentage of Spot Instances.

d. (Optional) When you specify a percentage for Spot Instances, select **Include On-Demand base capacity** and then specify the minimum amount of the Auto Scaling group's initial capacity that must be fulfilled by On-Demand Instances. Anything beyond the base capacity uses the **Instances distribution** settings to determine how many On-Demand Instances and Spot Instances to launch.

e. Under **Allocation strategies**, for **On-Demand allocation strategy**, choose an allocation strategy. When you manually choose your instance types, **Prioritized** is selected by default.

f. For **Spot allocation strategy**, choose an allocation strategy. **Price capacity optimized** is selected by default. **Lowest price** is hidden by default and only appears when you choose **Show all strategies**.

   - If you choose **Lowest price**, enter the number of lowest priced pools to diversify across for **Lowest priced pools**.
   - If you choose **Capacity optimized**, you can optionally check the **Prioritize instance types** box to let Amazon EC2 Auto Scaling choose which instance type to launch first based on the order your instance types are listed in.

g. For **Capacity Rebalancing**, choose whether to enable or disable Capacity Rebalancing. Use Capacity Rebalancing to automatically respond when your Spot Instances approach termination from a Spot interruption. For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

h. Under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch template.

i. For **Availability Zones and subnets**, choose one or more subnets in the specified VPC. Use subnets in multiple Availability Zones for high availability. For more information, see Considerations when choosing VPC subnets.

j. Choose **Next**, **Next**.

7. For the **Configure group size and scaling policies** step, do the following:

   a. Under **Group size**, for **Desired capacity**, enter the initial number of instances to launch.

      By default, the desired capacity is expressed as the number of instances. If you assigned weights to your instance types, you must convert this value to the same unit of measurement that you used to assign weights, such as the number of vCPUs.

   b. In the **Scaling** section, under **Scaling limits**, if your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed. For more information, see Set scaling limits for your Auto Scaling group.

8. Choose **Skip to review**.

9. On the **Review** page, choose **Create Auto Scaling group**.

Create a mixed instances group (AWS CLI)

To create a mixed instances group using the command line

Use one of the following commands:

- create-auto-scaling-group (AWS CLI)
- New-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Example configurations

The following example configurations show how to create mixed instances groups using different Spot allocation strategies.

> **Note**
>
> These examples show how to use a configuration file formatted in JSON or YAML. If you use AWS CLI version 1, you must specify a JSON-formatted configuration file. If you use AWS CLI version 2, you can specify a configuration file formatted in either YAML or JSON.

Examples

- Example 1: Launch Spot Instances using the capacity-optimized allocation strategy
- Example 2: Launch Spot Instances using the capacity-optimized-prioritized allocation strategy
- Example 3: Launch Spot Instances using the lowest-price allocation strategy diversified over two pools
- Example 4: Launch Spot Instances using the price-capacity-optimized allocation strategy

Example 1: Launch Spot Instances using the `capacity-optimized` allocation strategy

The following create-auto-scaling-group command creates an Auto Scaling group that specifies the following:

- The percentage of the group to launch as On-Demand Instances (`0`) and a base number of On-Demand Instances to start with (`1`).
- The instance types to launch in priority order (`c5.large`, `c5a.large`, `m5.large`, `m5a.large`, `c4.large`, `m4.large`, `c3.large`, `m3.large`) .
- The subnets in which to launch the instances (`subnet-5ea0c127`, `subnet-6194ea3b`, `subnet-c934b782`). Each corresponds to a different Availability Zone.
- The launch template (`my-launch-template`) and the launch template version (`$Default`).

When Amazon EC2 Auto Scaling attempts to fulfill your On-Demand capacity, it launches the `c5.large` instance type first. The Spot Instances come from the optimal Spot pool in each Availability Zone based on Spot Instance capacity.

JSON

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
    "AutoScalingGroupName": "my-asg",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Default"
```

```
            },
            "Overrides": [
                {
                    "InstanceType": "c5.large"
                },
                {
                    "InstanceType": "c5a.large"
                },
                {
                    "InstanceType": "m5.large"
                },
                {
                    "InstanceType": "m5a.large"
                },
                {
                    "InstanceType": "c4.large"
                },
                {
                    "InstanceType": "m4.large"
                },
                {
                    "InstanceType": "c3.large"
                },
                {
                    "InstanceType": "m3.large"
                }
            ]
        },
        "InstancesDistribution": {
            "OnDemandBaseCapacity": 1,
            "OnDemandPercentageAboveBaseCapacity": 0,
            "SpotAllocationStrategy": "capacity-optimized"
        }
    },
    "MinSize": 1,
    "MaxSize": 5,
    "DesiredCapacity": 3,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
}
```

YAML

Alternatively, you can use the following create-auto-scaling-group command to create the Auto Scaling group. This references a YAML file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-yaml file://~/config.yaml
```

The `config.yaml` file contains the following content.

```
---
AutoScalingGroupName: my-asg
MixedInstancesPolicy:
  LaunchTemplate:
    LaunchTemplateSpecification:
      LaunchTemplateName: my-launch-template
      Version: $Default
    Overrides:
    - InstanceType: c5.large
    - InstanceType: c5a.large
    - InstanceType: m5.large
    - InstanceType: m5a.large
    - InstanceType: c4.large
    - InstanceType: m4.large
    - InstanceType: c3.large
    - InstanceType: m3.large
  InstancesDistribution:
    OnDemandBaseCapacity: 1
    OnDemandPercentageAboveBaseCapacity: 0
    SpotAllocationStrategy: capacity-optimized
MinSize: 1
MaxSize: 5
DesiredCapacity: 3
VPCZoneIdentifier: subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782
```

Example 2: Launch Spot Instances using the `capacity-optimized-prioritized` allocation strategy

The following create-auto-scaling-group command creates an Auto Scaling group that specifies the following:

- The percentage of the group to launch as On-Demand Instances (`0`) and a base number of On-Demand Instances to start with (1).

- The instance types to launch in priority order (`c5.large`, `c5a.large`, `m5.large`, `m5a.large`, `c4.large`, `m4.large`, `c3.large`, `m3.large`) .
- The subnets in which to launch the instances (`subnet-5ea0c127`, `subnet-6194ea3b`, `subnet-c934b782`). Each corresponds to a different Availability Zone.
- The launch template (`my-launch-template`) and the launch template version (`$Latest`).

When Amazon EC2 Auto Scaling attempts to fulfill your On-Demand capacity, it launches the `c5.large` instance type first. When Amazon EC2 Auto Scaling attempts to fulfill your Spot capacity, it honors the instance type priorities on a best-effort basis. However, it optimizes for capacity first.

JSON

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
    "AutoScalingGroupName": "my-asg",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Latest"
            },
            "Overrides": [
                {
                    "InstanceType": "c5.large"
                },
                {
                    "InstanceType": "c5a.large"
                },
                {
                    "InstanceType": "m5.large"
                },
                {
                    "InstanceType": "m5a.large"
                },
                {
                    "InstanceType": "c4.large"
```

```
                },
                {
                    "InstanceType": "m4.large"
                },
                {
                    "InstanceType": "c3.large"
                },
                {
                    "InstanceType": "m3.large"
                }
            ]
        },
        "InstancesDistribution": {
            "OnDemandBaseCapacity": 1,
            "OnDemandPercentageAboveBaseCapacity": 0,
            "SpotAllocationStrategy": "capacity-optimized-prioritized"
        }
    },
    "MinSize": 1,
    "MaxSize": 5,
    "DesiredCapacity": 3,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
}
```

YAML

Alternatively, you can use the following create-auto-scaling-group command to create the Auto Scaling group. This references a YAML file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-yaml file://~/config.yaml
```

The `config.yaml` file contains the following content.

```
---
AutoScalingGroupName: my-asg
MixedInstancesPolicy:
  LaunchTemplate:
    LaunchTemplateSpecification:
      LaunchTemplateName: my-launch-template
      Version: $Default
    Overrides:
    - InstanceType: c5.large
    - InstanceType: c5a.large
```

```
    - InstanceType: m5.large
    - InstanceType: m5a.large
    - InstanceType: c4.large
    - InstanceType: m4.large
    - InstanceType: c3.large
    - InstanceType: m3.large
  InstancesDistribution:
    OnDemandBaseCapacity: 1
    OnDemandPercentageAboveBaseCapacity: 0
    SpotAllocationStrategy: capacity-optimized-prioritized
MinSize: 1
MaxSize: 5
DesiredCapacity: 3
VPCZoneIdentifier: subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782
```

Example 3: Launch Spot Instances using the `lowest-price` allocation strategy diversified over two pools

The following create-auto-scaling-group command creates an Auto Scaling group that specifies the following:

- The percentage of the group to launch as On-Demand Instances (`50`). (This does not specify a base number of On-Demand Instances to start with.)
- The instance types to launch in priority order (`c5.large`, `c5a.large`, `m5.large`, `m5a.large`, `c4.large`, `m4.large`, `c3.large`, `m3.large`).
- The subnets in which to launch the instances (`subnet-5ea0c127`, `subnet-6194ea3b`, `subnet-c934b782`). Each corresponds to a different Availability Zone.
- The launch template (`my-launch-template`) and the launch template version (`$Latest`).

When Amazon EC2 Auto Scaling attempts to fulfill your On-Demand capacity, it launches the `c5.large` instance type first. For your Spot capacity, Amazon EC2 Auto Scaling attempts to launch the Spot Instances evenly across the two lowest priced pools in each Availability Zone.

JSON

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
```

```
    "AutoScalingGroupName": "my-asg",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Latest"
            },
            "Overrides": [
                {
                    "InstanceType": "c5.large"
                },
                {
                    "InstanceType": "c5a.large"
                },
                {
                    "InstanceType": "m5.large"
                },
                {
                    "InstanceType": "m5a.large"
                },
                {
                    "InstanceType": "c4.large"
                },
                {
                    "InstanceType": "m4.large"
                },
                {
                    "InstanceType": "c3.large"
                },
                {
                    "InstanceType": "m3.large"
                }
            ]
        },
        "InstancesDistribution": {
            "OnDemandPercentageAboveBaseCapacity": 50,
            "SpotAllocationStrategy": "lowest-price",
            "SpotInstancePools": 2
        }
    },
    "MinSize": 1,
    "MaxSize": 5,
    "DesiredCapacity": 3,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
```

```
}
```

YAML

Alternatively, you can use the following create-auto-scaling-group command to create the Auto Scaling group. This references a YAML file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-yaml file://~/config.yaml
```

The `config.yaml` file contains the following content.

```
---
AutoScalingGroupName: my-asg
MixedInstancesPolicy:
  LaunchTemplate:
    LaunchTemplateSpecification:
      LaunchTemplateName: my-launch-template
      Version: $Default
    Overrides:
    - InstanceType: c5.large
    - InstanceType: c5a.large
    - InstanceType: m5.large
    - InstanceType: m5a.large
    - InstanceType: c4.large
    - InstanceType: m4.large
    - InstanceType: c3.large
    - InstanceType: m3.large
  InstancesDistribution:
    OnDemandPercentageAboveBaseCapacity: 50
    SpotAllocationStrategy: lowest-price
    SpotInstancePools: 2
MinSize: 1
MaxSize: 5
DesiredCapacity: 3
VPCZoneIdentifier: subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782
```

Example 4: Launch Spot Instances using the `price-capacity-optimized` allocation strategy

The following create-auto-scaling-group command creates an Auto Scaling group that specifies the following:

- The percentage of the group to launch as On-Demand Instances (`30`). (This does not specify a base number of On-Demand Instances to start with.)
- The instance types to launch in priority order (`c5.large`, `c5a.large`, `m5.large`, `m5a.large`, `c4.large`, `m4.large`, `c3.large`, `m3.large`).
- The subnets in which to launch the instances (`subnet-5ea0c127`, `subnet-6194ea3b`, `subnet-c934b782`). Each corresponds to a different Availability Zone.
- The launch template (`my-launch-template`) and the launch template version (`$Latest`).

When Amazon EC2 Auto Scaling attempts to fulfill your On-Demand capacity, it launches the `c5.large` instance type first. For your Spot capacity, Amazon EC2 Auto Scaling attempts to launch the Spot Instances from Spot Instance pools with the lowest price possible, but also with optimal capacity for the number of instances that are launching.

JSON

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
    "AutoScalingGroupName": "my-asg",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Latest"
            },
            "Overrides": [
                {
                    "InstanceType": "c5.large"
                },
                {
                    "InstanceType": "c5a.large"
                },
                {
                    "InstanceType": "m5.large"
                },
                {
                    "InstanceType": "m5a.large"
```

```
                },
                {
                    "InstanceType": "c4.large"
                },
                {
                    "InstanceType": "m4.large"
                },
                {
                    "InstanceType": "c3.large"
                },
                {
                    "InstanceType": "m3.large"
                }
            ]
        },
        "InstancesDistribution": {
            "OnDemandPercentageAboveBaseCapacity": 30,
            "SpotAllocationStrategy": "price-capacity-optimized"
        }
    },
    "MinSize": 1,
    "MaxSize": 5,
    "DesiredCapacity": 3,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
}
```

YAML

Alternatively, you can use the following create-auto-scaling-group command to create the Auto Scaling group. This references a YAML file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-yaml file://~/config.yaml
```

The `config.yaml` file contains the following content.

```
---
AutoScalingGroupName: my-asg
MixedInstancesPolicy:
  LaunchTemplate:
    LaunchTemplateSpecification:
      LaunchTemplateName: my-launch-template
      Version: $Default
```

```
    Overrides:
    - InstanceType: c5.large
    - InstanceType: c5a.large
    - InstanceType: m5.large
    - InstanceType: m5a.large
    - InstanceType: c4.large
    - InstanceType: m4.large
    - InstanceType: c3.large
    - InstanceType: m3.large
  InstancesDistribution:
    OnDemandPercentageAboveBaseCapacity: 30
    SpotAllocationStrategy: price-capacity-optimized
MinSize: 1
MaxSize: 5
DesiredCapacity: 3
VPCZoneIdentifier: subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782
```

Configure an Auto Scaling group to use instance weights

When you use multiple instance types, you can specify how many units to associate with each instance type, and then specify the capacity of your group with the same unit of measurement. This capacity specification option is known as weights.

For example, let's say that you run a compute-intensive application that performs best with at least 8 vCPUs and 15 GiB of RAM. If you use `c5.2xlarge` as your base unit, any of the following EC2 instance types would meet your application needs.

Instance types example

Instance type	vCPU	Memory (GiB)
`c5.2xlarge`	8	16
`c5.4xlarge`	16	32
`c5.12xlarge`	48	96
`c5.18xlarge`	72	144
`c5.24xlarge`	96	192

By default, all instance types have equal weight regardless of size. In other words, whether Amazon EC2 Auto Scaling launches a large or small instance type, each instance counts the same toward the desired capacity of the Auto Scaling group.

With weights, however, you assign a number value that specifies how many units to associate with each instance type. For example, if the instances are of different sizes, a `c5.2xlarge` instance could have the weight of 2, and a `c5.4xlarge` (which is two times bigger) could have the weight of 4, and so on. Then, when Amazon EC2 Auto Scaling scales the group, these weights translate into the number of units that each instance counts toward your desired capacity.

The weights do not change which instance types Amazon EC2 Auto Scaling chooses to launch; instead, the allocation strategies do that. For more information, see Allocation strategies for multiple instance types.

> **Important**
>
> To configure an Auto Scaling group to fulfill its desired capacity using the number of vCPUs or the amount of memory of each instance type, we recommend using attribute-based instance type selection. Setting the `DesiredCapacityType` parameter automatically specifies the number of units to associate with each instance type based on the value that you set for this parameter. For more information, see Create mixed instances group using attribute-based instance type selection.

Contents

- Considerations
- Instance weight behaviors
- Configure an Auto Scaling group to use weights
- Spot price per unit hour example

Considerations

This section discusses key considerations for effectively implementing weights.

- Choose a few instance types that match your application's performance needs. Decide the weight each instance type should count toward the desired capacity of your Auto Scaling group based on its capabilities. These weights apply to current and future instances.

- Avoid large ranges between weights. For example, don't specify a weight of 1 for an instance type when the next larger instance type has a weight of 200. The difference between the smallest and largest weights shouldn't be extreme, either. Extreme weight differences can negatively impact cost-performance optimization.
- Specify the group's desired capacity in units, not instances. For example, if you use vCPU-based weights, set your desired number of cores and also the minimum and maximum.
- Set your weights and desired capacity so that the desired capacity is at least two to three times larger than your largest weight.

Note the following when updating existing groups:

- When you add weights to an existing group, include weights for all instance types currently in use.
- When you add or change weights, Amazon EC2 Auto Scaling will launch or terminate instances to reach the desired capacity based on the new weight values.
- If you remove an instance type, running instances of that type keep their last weight, even if no longer defined.

Instance weight behaviors

When you use instance weights, Amazon EC2 Auto Scaling behaves in the following way:

- Current capacity will either be at the desired capacity or above it. Current capacity can exceed the desired capacity if instances launched that exceed the remaining desired capacity units. For example, suppose that you specify two instance types, `c5.2xlarge` and `c5.12xlarge`, and you assign instance weights of 2 for `c5.2xlarge` and 12 for `c5.12xlarge`. If there are five units remaining to fulfill the desired capacity, and Amazon EC2 Auto Scaling provisions a `c5.12xlarge`, the desired capacity is exceeded by seven units.
- When launching instances, Amazon EC2 Auto Scaling prioritizes distributing capacity across Availability Zones and respecting allocation strategies over exceeding the desired capacity.
- Amazon EC2 Auto Scaling can exceed the maximum capacity limit to maintain balance across Availability Zones, using your preferred allocation strategies. The hard limit enforced by Amazon EC2 Auto Scaling is your desired capacity plus your largest weight.

Configure an Auto Scaling group to use weights

You can configure an Auto Scaling group to use weights, as shown in the following AWS CLI examples. For instructions on using the console, see Create a mixed instances group by manually choosing instance types.

To configure a new Auto Scaling group to use weights (AWS CLI)

Use the create-auto-scaling-group command. For example, the following command creates a new Auto Scaling group and assigns weights by specifying the following:

- The percentage of the group to launch as On-Demand Instances (`0`)
- The allocation strategy for Spot Instances in each Availability Zone (`capacity-optimized`)
- The instance types to launch in priority order (`m4.16xlarge`, `m5.24xlarge`)
- The instance weights that correspond to the relative size difference (vCPUs) between instance types (16, 24)
- The subnets in which to launch the instances (`subnet-5ea0c127`, `subnet-6194ea3b`, `subnet-c934b782`), each corresponding to a different Availability Zone
- The launch template (`my-launch-template`) and the launch template version (`$Latest`)

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
    "AutoScalingGroupName": "my-asg",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Latest"
            },
            "Overrides": [
                {
                    "InstanceType": "m4.16xlarge",
                    "WeightedCapacity": "16"
                },
                {
```

```
                    "InstanceType": "m5.24xlarge",
                    "WeightedCapacity": "24"
                }
            ]
        },
        "InstancesDistribution": {
            "OnDemandPercentageAboveBaseCapacity": 0,
            "SpotAllocationStrategy": "capacity-optimized"
        }
    },
    "MinSize": 160,
    "MaxSize": 720,
    "DesiredCapacity": 480,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782",
    "Tags": []
}
```

To configure an existing Auto Scaling group to use weights (AWS CLI)

Use the update-auto-scaling-group command. For example, the following command assigns weights to instance types in an existing Auto Scaling group by specifying the following:

- The instance types to launch in priority order (`c5.18xlarge`, `c5.24xlarge`, `c5.2xlarge`, `c5.4xlarge`)
- The instance weights that correspond to the relative size difference (vCPUs) between instance types (18, 24, 2, 4)
- The new, increased desired capacity, which is larger than the largest weight

```
aws autoscaling update-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
    "AutoScalingGroupName": "my-existing-asg",
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "Overrides": [
                {
                    "InstanceType": "c5.18xlarge",
                    "WeightedCapacity": "18"
```

```
                },
                {
                    "InstanceType": "c5.24xlarge",
                    "WeightedCapacity": "24"
                },
                {
                    "InstanceType": "c5.2xlarge",
                    "WeightedCapacity": "2"
                },
                {
                    "InstanceType": "c5.4xlarge",
                    "WeightedCapacity": "4"
                }
            ]
        }
    },
    "MinSize": 0,
    "MaxSize": 100,
    "DesiredCapacity": 100
}
```

To verify the weights using the command line

Use one of the following commands:

- describe-auto-scaling-groups (AWS CLI)
- Get-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Spot price per unit hour example

The following table compares the hourly price for Spot Instances in different Availability Zones in US East (Northern Virginia) with the price for On-Demand Instances in the same Region. The prices shown are example pricing and not current pricing. These are your costs per *instance* hour.

Example: Spot pricing per instance hour

Instance type	us-east-1a	us-east-1b	us-east-1c	On-Demand pricing
`c5.2xlarge`	$0.180	$0.191	$0.170	$0.34

Instance type	us-east-1a	us-east-1b	us-east-1c	On-Demand pricing
`c5.4xlarge`	$0.341	$0.361	$0.318	$0.68
`c5.12xlarge`	$0.779	$0.777	$0.777	$2.04
`c5.18xlarge`	$1.207	$1.475	$1.357	$3.06
`c5.24xlarge`	$1.555	$1.555	$1.555	$4.08

With instance weights, you can evaluate your costs based on what you use per *unit* hour. You can determine the price per unit hour by dividing your price for an instance type by the number of units that it represents. For On-Demand Instances, the price per unit hour is the same when deploying one instance type as it is when deploying a different size of the same instance type. In contrast, however, the Spot price per unit hour varies by Spot pool.

The following example shows how the Spot price per unit hour calculation works with instance weights. For ease of calculation, let's say you want to launch Spot Instances only in `us-east-1a`. The per unit hour price is captured in the following table.

Example: Spot Price per unit hour example

Instance type	us-east-1a	Instance weight	Price per unit hour
`c5.2xlarge`	$0.180	2	$0.090
`c5.4xlarge`	$0.341	4	$0.085
`c5.12xlarge`	$0.779	12	$0.065
`c5.18xlarge`	$1.207	18	$0.067
`c5.24xlarge`	$1.555	24	$0.065

Use a different launch template for an instance type

In addition to using multiple instance types, you can also use multiple launch templates.

For example, say that you configure an Auto Scaling group for compute-intensive applications and want to include a mix of C5, C5a, and C6g instance types. However, C6g instances feature an AWS Graviton processor based on 64-bit Arm architecture, while the C5 and C5a instances run on 64-bit Intel x86 processors. The AMIs for C5 and C5a instances both work on each of those instances, but not on C6g instances. To solve this problem, use a different launch template for C6g instances. You can still use the same launch template for C5 and C5a instances.

This section contains procedures for using the AWS CLI to perform tasks related to using multiple launch templates. Currently, this feature is available only if you use the AWS CLI or an SDK, and is not available from the console.

Contents

- Configure an Auto Scaling group to use multiple launch templates
- Related resources

Configure an Auto Scaling group to use multiple launch templates

You can configure an Auto Scaling group to use multiple launch templates, as shown in the following examples.

To configure a new Auto Scaling group to use multiple launch templates (AWS CLI)

Use the create-auto-scaling-group command. For example, the following command creates a new Auto Scaling group. It specifies the `c5.large`, `c5a.large`, and `c6g.large` instance types and defines a new launch template for the `c6g.large` instance type to ensure that an appropriate AMI is used to launch Arm instances. Amazon EC2 Auto Scaling uses the order of instance types to determine which instance type to use first when fulfilling On-Demand capacity.

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
  "AutoScalingGroupName":"my-asg",
  "MixedInstancesPolicy":{
```

```
    "LaunchTemplate":{
      "LaunchTemplateSpecification":{
        "LaunchTemplateName":"my-launch-template-for-x86",
        "Version":"$Latest"
      },
      "Overrides":[
        {
          "InstanceType":"c6g.large",
          "LaunchTemplateSpecification": {
            "LaunchTemplateName": "my-launch-template-for-arm",
            "Version": "$Latest"
          }
        },
        {
          "InstanceType":"c5.large"
        },
        {
          "InstanceType":"c5a.large"
        }
      ]
    },
    "InstancesDistribution":{
      "OnDemandBaseCapacity": 1,
      "OnDemandPercentageAboveBaseCapacity": 50,
      "SpotAllocationStrategy": "capacity-optimized"
    }
  },
  "MinSize":1,
  "MaxSize":5,
  "DesiredCapacity":3,
  "VPCZoneIdentifier":"subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782",
  "Tags":[ ]
}
```

To configure an existing Auto Scaling group to use multiple launch templates (AWS CLI)

Use the update-auto-scaling-group command. For example, the following command assigns the launch template named *my-launch-template-for-arm* to the *c6g.large* instance type for the Auto Scaling group named *my-asg*.

```
aws autoscaling update-auto-scaling-group --cli-input-json file://~/config.json
```

The `config.json` file contains the following content.

```
{
  "AutoScalingGroupName":"my-asg",
  "MixedInstancesPolicy":{
    "LaunchTemplate":{
      "Overrides":[
        {
          "InstanceType":"c6g.large",
          "LaunchTemplateSpecification": {
            "LaunchTemplateName": "my-launch-template-for-arm",
            "Version": "$Latest"
          }
        },
        {
          "InstanceType":"c5.large"
        },
        {
          "InstanceType":"c5a.large"
        }
      ]
    }
  }
}
```

To verify the launch templates for an Auto Scaling group

Use one of the following commands:

- describe-auto-scaling-groups (AWS CLI)
- Get-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Related resources

You can find an example of specifying multiple launch templates using attribute-based instance type selection in a AWS CloudFormation template on AWS re:Post.

Create Auto Scaling groups using launch configurations

> **Important**
>
> You can not call `CreateLaunchConfiguration` with new Amazon EC2 instance types that are released after **December 31, 2022**. In addition, any new accounts created on or

after **June 1, 2023** will not have the option to create new launch configurations through the console. Starting on **October 1, 2024**, new accounts will not be able to create new launch configurations by using the console, API, CLI, and CloudFormation. Migrate to launch templates to ensure that you don't need to create new launch configurations now or in the future. For information about migrating your Auto Scaling groups to launch templates, see Migrate your Auto Scaling groups to launch templates.

If you have created a launch configuration or an EC2 instance, you can create an Auto Scaling group that uses a launch configuration as a configuration template for its EC2 instances. The launch configuration specifies information such as the AMI ID, instance type, key pair, security groups, and block device mapping for your instances. For information about creating launch configurations, see Create a launch configuration.

You must have sufficient permissions to create an Auto Scaling group. You must also have sufficient permissions to create the service-linked role that Amazon EC2 Auto Scaling uses to perform actions on your behalf if it does not yet exist. For examples of IAM policies that an administrator can use as a reference for granting you permissions, see Identity-based policy examples.

Contents

- Create an Auto Scaling group using a launch configuration
- Create an Auto Scaling group using parameters from an existing instance

Create an Auto Scaling group using a launch configuration

Important

We provide information about launch configurations for customers who have not yet migrated from launch configurations to launch templates. For information about migrating your Auto Scaling groups to launch templates, see Migrate your Auto Scaling groups to launch templates.

When you create an Auto Scaling group, you must specify the necessary information to configure the Amazon EC2 instances, the Availability Zones and VPC subnets for the instances, the desired capacity, and the minimum and maximum capacity limits.

The following procedure demonstrates how to create an Auto Scaling group using a launch configuration. You cannot modify a launch configuration after it is created, but you can replace the launch configuration for an Auto Scaling group. For more information, see Change the launch configuration for an Auto Scaling group.

Prerequisites

- You must have created a launch configuration. For more information, see Create a launch configuration.

To create an Auto Scaling group using a launch configuration (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the same AWS Region that you used when you created the launch configuration.
3. Choose **Create an Auto Scaling group**.
4. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter a name for your Auto Scaling group.
5. To choose a launch configuration, do the following:

   a. For **Launch template**, choose **Switch to launch configuration**.

   b. For **Launch configuration**, choose an existing launch configuration.

   c. Verify that your launch configuration supports all of the options that you are planning to use, and then choose **Next**.
6. On the **Configure instance launch options** page, under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch configuration.
7. For **Availability Zones and subnets**, choose one or more subnets in the specified VPC. Use subnets in multiple Availability Zones for high availability. For more information, see Considerations when choosing VPC subnets.
8. Choose **Next**.

   Or, you can accept the rest of the defaults, and choose **Skip to review**.
9. (Optional) On the **Configure advanced options** page, configure the following options, and then choose **Next**:

a. (Optional) For **Health checks**, **Additional health check types**, select **Turn on Amazon EBS health checks**. For more information, see Monitor Auto Scaling instances with impaired Amazon EBS volumes using health checks.

b. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.

c. Under **Additional settings**, **Monitoring**, choose whether to enable CloudWatch group metrics collection. These metrics provide measurements that can be indicators of a potential issue, such as number of terminating instances or number of pending instances. For more information, see Monitor CloudWatch metrics for your Auto Scaling groups and instances.

d. For **Enable default instance warmup**, select this option and choose the warmup time for your application. If you are creating an Auto Scaling group that has a scaling policy, the default instance warmup feature improves the Amazon CloudWatch metrics used for dynamic scaling. For more information, see Set the default instance warmup for an Auto Scaling group.

10. (Optional) On the **Configure group size and scaling policies** page, configure the following options, and then choose **Next**:

    a. Under **Group size**, for **Desired capacity**, enter the initial number of instances to launch.

    b. In the **Scaling** section, under **Scaling limits**, if your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed. For more information, see Set scaling limits for your Auto Scaling group.

    c. For **Automatic scaling**, choose whether you want to create a target tracking scaling policy. You can also create this policy after your create your Auto Scaling group.

       If you choose **Target tracking scaling policy**, follow the directions in Create a target tracking scaling policy to create the policy.

    d. For **Instance maintenance policy**, choose whether you want to create an instance maintenance policy. You can also create this policy after your create your Auto Scaling group. Follow the directions in Set an instance maintenance policy to create the policy.

   e. Under **Instance scale-in protection**, choose whether to enable instance scale-in protection. For more information, see Use instance scale-in protection to control instance termination.

11. (Optional) To receive notifications, for **Add notification**, configure the notification, and then choose **Next**. For more information, see Amazon SNS notification options for Amazon EC2 Auto Scaling.
12. (Optional) To add tags, choose **Add tag**, provide a tag key and value for each tag, and then choose **Next**. For more information, see Tag Auto Scaling groups and instances.
13. On the **Review** page, choose **Create Auto Scaling group**.

To create an Auto Scaling group using the command line

You can use one of the following commands:

- create-auto-scaling-group (AWS CLI)
- New-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Create an Auto Scaling group using parameters from an existing instance

> **Important**
>
> We provide information about launch configurations for customers who have not yet migrated from launch configurations to launch templates. For information about migrating your Auto Scaling groups to launch templates, see Migrate your Auto Scaling groups to launch templates.

If this is your first time creating an Auto Scaling group, we recommend you use the console to create a launch template from an existing EC2 instance. Then use the launch template to create a new Auto Scaling group. For this procedure, see Create an Auto Scaling group using the Amazon EC2 launch wizard.

The following procedure shows how to create an Auto Scaling group by specifying an existing instance to use as a base for launching other instances. Multiple parameters are required to create

an EC2 instance, such as the Amazon Machine Image (AMI) ID, instance type, key pair, and security group. All of this information is also used by Amazon EC2 Auto Scaling to launch instances on your behalf when there is a need to scale. This information is stored in either a launch template or a launch configuration.

When you use an existing instance, Amazon EC2 Auto Scaling creates an Auto Scaling group that launches instances based on a launch configuration that's created at the same time. The new launch configuration has the same name as the Auto Scaling group, and it includes certain configuration details from the identified instance.

The following configuration details are copied from the identified instance into the launch configuration:

- AMI ID
- Instance type
- Key pair
- Security groups
- IP address type (public or private)
- IAM instance profile, if applicable
- Monitoring (true or false)
- EBS optimized (true or false)
- Tenancy setting, if launching into a VPC (shared or dedicated)
- Kernel ID and RAM disk ID, if applicable
- User data, if specified
- Spot (maximum) price

The VPC subnet and Availability Zone are copied from the identified instance to the Auto Scaling group's own resource definition.

If the identified instance is in a placement group, the new Auto Scaling group launches instances into the same placement group as the identified instance. Because the launch configuration settings do not allow a placement group to be specified, the placement group is copied to the `PlacementGroup` attribute of the new Auto Scaling group.

The following configuration details are not copied from your identified instance:

- Storage: The block devices (EBS volumes and instance store volumes) are not copied from the identified instance. Instead, the block device mapping created as part of creating the AMI determines which devices are used.
- Number of network interfaces: The network interfaces are not copied from your identified instance. Instead, Amazon EC2 Auto Scaling uses its default settings to create one network interface, which is the primary network interface (eth0).
- Instance metadata options: The metadata accessible, metadata version, and token response hop limit settings are not copied from the identified instance. Instead, Amazon EC2 Auto Scaling uses its default settings. For more information, see Configure the instance metadata options.
- Load balancers: If the identified instance is registered with one or more load balancers, the information about the load balancer is not copied to the load balancer or target group attribute of the new Auto Scaling group.
- Tags: If the identified instance has tags, the tags are not copied to the `Tags` attribute of the new Auto Scaling group.

Prerequisites

The EC2 instance must meet the following criteria:

- The instance is not a member of another Auto Scaling group.
- The instance is in the `running` state.
- The AMI that was used to launch the instance must still exist.

Create an Auto Scaling group from an EC2 instance (console)

To create an Auto Scaling group from an EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Instances**, choose **Instances**, and then select an instance.
3. Choose **Actions**, **Instance settings**, **Attach to Auto Scaling Group**.
4. On the **Attach to Auto Scaling group** page, for **Auto Scaling Group**, enter a name for the group, and then choose **Attach**.

   After the instance is attached, it is considered part of the Auto Scaling group. The new Auto Scaling group is created using a new launch configuration with the same name that you

specified for the Auto Scaling group. The Auto Scaling group has a desired capacity and maximum size of 1.

5. (Optional) To edit the settings for the Auto Scaling group, on the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**. Select the check box next to the new Auto Scaling group, choose the **Edit** button that is above the list of groups, change the settings as needed, and then choose **Update**.

Create an Auto Scaling group from an EC2 instance (AWS CLI)

The following procedure shows you how to use a CLI command to create an Auto Scaling group from an EC2 instance.

This procedure does not add the instance to the Auto Scaling group. For the instance to be attached, you must run the attach-instances command after the Auto Scaling group has been created.

Before you begin, find the ID of the EC2 instance using the Amazon EC2 console or the describe-instances command.

To use your current instance as a template

- Use the following create-auto-scaling-group command to create an Auto Scaling group, `my-asg-from-instance`, from the EC2 instance `i-0e69cc3f05f825f4f`.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg-from-instance \
  --instance-id i-0e69cc3f05f825f4f --min-size 1 --max-size 2 --desired-capacity 2
```

To verify that your Auto Scaling group has launched instances

- Use the following describe-auto-scaling-groups command to verify that the Auto Scaling group was created successfully.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg-from-instance
```

The following example response shows that the desired capacity of the group is 2, the group has 2 running instances, and the launch configuration is named `my-asg-from-instance`.

```
{
  "AutoScalingGroups":[
    {
      "AutoScalingGroupName":"my-asg-from-instance",
      "AutoScalingGroupARN":"arn",
      "LaunchConfigurationName":"my-asg-from-instance",
      "MinSize":1,
      "MaxSize":2,
      "DesiredCapacity":2,
      "DefaultCooldown":300,
      "AvailabilityZones":[
        "us-west-2a"
      ],
      "LoadBalancerNames":[],
      "TargetGroupARNs":[],
      "HealthCheckType":"EC2",
      "HealthCheckGracePeriod":0,
      "Instances":[
        {
          "InstanceId":"i-06905f55584de02da",
          "InstanceType":"t2.micro",
          "AvailabilityZone":"us-west-2a",
          "LifecycleState":"InService",
          "HealthStatus":"Healthy",
          "LaunchConfigurationName":"my-asg-from-instance",
          "ProtectedFromScaleIn":false
        },
        {
          "InstanceId":"i-087b42219468eacde",
          "InstanceType":"t2.micro",
          "AvailabilityZone":"us-west-2a",
          "LifecycleState":"InService",
          "HealthStatus":"Healthy",
          "LaunchConfigurationName":"my-asg-from-instance",
          "ProtectedFromScaleIn":false
        }
      ],
      "CreatedTime":"2020-10-28T02:39:22.152Z",
      "SuspendedProcesses":[ ],
      "VPCZoneIdentifier":"subnet-6bea5f06",
      "EnabledMetrics":[ ],
      "Tags":[ ],
      "TerminationPolicies":[
```

```
        "Default"
      ],
      "NewInstancesProtectedFromScaleIn":false,
      "ServiceLinkedRoleARN":"arn",
      "TrafficSources":[]
    }
  ]
}
```

To view the launch configuration

- Use the following describe-launch-configurations command to view the details of the launch configuration.

```
aws autoscaling describe-launch-configurations --launch-configuration-names my-asg-from-instance
```

The following is example output:

```
{
  "LaunchConfigurations":[
    {
      "LaunchConfigurationName":"my-asg-from-instance",
      "LaunchConfigurationARN":"arn",
      "ImageId":"ami-0528a5175983e7f28",
      "KeyName":"my-key-pair-uswest2",
      "SecurityGroups":[
        "sg-05eaec502fcdadc2e"
      ],
      "ClassicLinkVPCSecurityGroups":[ ],
      "UserData":"",
      "InstanceType":"t2.micro",
      "KernelId":"",
      "RamdiskId":"",
      "BlockDeviceMappings":[ ],
      "InstanceMonitoring":{
        "Enabled":true
      },
      "CreatedTime":"2020-10-28T02:39:22.321Z",
      "EbsOptimized":false,
      "AssociatePublicIpAddress":true
```

```
    }
  ]
}
```

To terminate the instance

- If you no longer need the instance, you can terminate it. The following terminate-instances command terminates the instance `i-0e69cc3f05f825f4f`.

  ```
  aws ec2 terminate-instances --instance-ids i-0e69cc3f05f825f4f
  ```

  After you terminate an Amazon EC2 instance, you can't restart the instance. After termination, its data is gone and the volume can't be attached to any instance. To learn more about terminating instances, see Terminate an instance in the *Amazon EC2 User Guide*.

Update an Auto Scaling group

You can update most of your Auto Scaling group's details. You can't update the name of an Auto Scaling group or change its AWS Region.

To update an Auto Scaling group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Choose your Auto Scaling group to display information about the group, with tabs for **Details**, **Activity**, **Automatic scaling**, **Instance management**, **Monitoring**, and **Instance refresh**.
3. Choose the tabs for the configuration areas that you're interested in and update the settings as needed. For each setting that you edit, choose **Update** to save your changes to the Auto Scaling group's configuration.

   - **Details** tab

     These are the general settings for your Auto Scaling group. You can edit and manage these in the same way as during Auto Scaling group creation.

     The **Advanced configurations** section has some options that are not available when creating the group such as termination policies, cooldown, suspended processes, and maximum

instance lifetime. You can also view, but not edit the placement group and service-linked role of the Auto Scaling group.

If the group is associated with Elastic Load Balancing resources, see Add an Availability Zone before changing Availability Zones. Some restrictions on the load balancer might prevent you from applying changes to your group's Availability Zones to your load balancer's Availability Zones.

- **Activity** tab
  - **Activity notifications** – Amazon SNS notifications
- **Automatic scaling** tab
  - **Dynamic scaling policies** – Dynamic scaling policies
  - **Predictive scaling policies** – Predictive scaling policies
  - **Scheduled actions** – Scheduled actions
- **Instance management** tab
  - **Lifecycle hooks** – Lifecycle hooks
  - **Warm pool** – Warm pools
- **Monitoring** tab
  - There is just a single option in this tab, which lets you enable or disable CloudWatch group metrics collection.

To update an Auto Scaling group using the command line

You can use one of the following commands:

- update-auto-scaling-group (AWS CLI)
- Update-ASAutoScalingGroup (AWS Tools for Windows PowerShell)

Update Auto Scaling instances

If you associate a new launch template or launch configuration with an Auto Scaling group, all new instances will get the updated configuration. Existing instances continue to run with the configuration that they were originally launched with. To apply your changes to existing instances, you have the following options:

- Start an instance refresh to replace the older instances. For more information, see Use an instance refresh to update instances in an Auto Scaling group.
- Wait for scaling activities to gradually replace older instances with newer instances based on your termination policies.
- Manually terminate them so that they are replaced by your Auto Scaling group.

> **Note**
>
> You can change the following instance attributes by specifying them as part of the launch template or launch configuration:
>
> - Amazon Machine Image (AMI)
> - block devices
> - key pair
> - instance type
> - security groups
> - user data
> - monitoring
> - IAM instance profile
> - placement tenancy
> - kernel
> - ramdisk
> - whether the instance has a public IP address

Tag Auto Scaling groups and instances

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

- A tag key (for example, `costcenter`, `environment`, or `project`)
- An optional field known as a tag value (for example, `111122223333` or `production`)

Tags help you do the following:

- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Using cost allocation tags in the *AWS Billing User Guide*.
- Control access to Auto Scaling groups based on tags. You can use conditions in your IAM policies to control access to Auto Scaling groups based on the tags on that group. For more information, see Tags for security.
- Filter and search for Auto Scaling groups based on the tags that you add. For more information, see Use tags to filter Auto Scaling groups.
- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related.

You can tag new or existing Auto Scaling groups. You can also propagate tags from an Auto Scaling group to the EC2 instances that it launches.

Tags are not propagated to Amazon EBS volumes. To add tags to Amazon EBS volumes, specify the tags in a launch template. For more information, see Create a launch template for an Auto Scaling group.

You can create and manage tags through the AWS Management Console, AWS CLI, or SDKs.

Contents

- Tag naming and usage restrictions
- EC2 instance tagging lifecycle
- Tag your Auto Scaling groups
- Delete tags
- Tags for security
- Control access to tags
- Use tags to filter Auto Scaling groups

Tag naming and usage restrictions

The following basic restrictions apply to tags:

- The maximum number of tags per resource is 50.
- The maximum number of tags that you can add or remove using a single call is 25.

- The maximum key length is 128 Unicode characters.
- The maximum value length is 256 Unicode characters.
- Tag keys and values are case-sensitive. As a best practice, decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types.
- Do not use the `aws:` prefix in your tag names or values, because it is reserved for AWS use. You can't edit or delete tag names or values with this prefix, and they do not count toward your tags per resource quota.

EC2 instance tagging lifecycle

If you have opted to propagate tags to your EC2 instances, the tags are managed as follows:

- When an Auto Scaling group launches instances, it adds tags to the instances during resource creation rather than after the resource is created.
- The Auto Scaling group automatically adds a tag to instances with a key of `aws:autoscaling:groupName` and a value of the Auto Scaling group name.
- If you specify instance tags in your launch template and you opted to propagate your group's tags to its instances, all the tags are merged. If the same tag key is specified for a tag in your launch template and a tag in your Auto Scaling group, then the tag value from the group takes precedence.
- When you attach existing instances, the Auto Scaling group adds the tags to the instances, overwriting any existing tags with the same tag key. It also adds a tag with a key of `aws:autoscaling:groupName` and a value of the Auto Scaling group name.
- When you detach an instance from an Auto Scaling group, it removes only the `aws:autoscaling:groupName` tag.

Tag your Auto Scaling groups

When you add a tag to your Auto Scaling group, you can specify whether it should be added to instances launched in the Auto Scaling group. If you modify a tag, the updated version of the tag is added to instances launched in the Auto Scaling group after the change. If you create or modify a tag for an Auto Scaling group, these changes are not made to instances that are already running in the Auto Scaling group.

Contents

- Add or modify tags (console)
- Add or modify tags (AWS CLI)

Add or modify tags (console)

To tag an Auto Scaling group on creation

When you use the Amazon EC2 console to create an Auto Scaling group, you can specify tag keys and values on the **Add tags** page of the Create Auto Scaling group wizard. To propagate a tag to the instances launched in the Auto Scaling group, make sure that you keep the **Tag new instances** option for that tag selected. Otherwise, you can deselect it.

To add or modify tags for an existing Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Tags**, **Edit**.
4. To modify existing tags, edit **Key** and **Value**.
5. To add a new tag, choose **Add tag** and edit **Key** and **Value**. You can keep **Tag new instances** selected to add the tag to the instances launched in the Auto Scaling group automatically, and deselect it otherwise.
6. When you have finished adding tags, choose **Update**.

Add or modify tags (AWS CLI)

The following examples show how to use the AWS CLI to add tags when you create Auto Scaling groups, and to add or modify tags for existing Auto Scaling groups.

To tag an Auto Scaling group on creation

Use the create-auto-scaling-group command to create a new Auto Scaling group and add a tag, for example, **`environment=production`**, to the Auto Scaling group. The tag is also added to any instances launched in the Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-configuration-name my-launch-config --min-size 1 --max-size 3 \
  --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782" \
  --tags Key=environment,Value=production,PropagateAtLaunch=true
```

To create or modify tags for an existing Auto Scaling group

Use the create-or-update-tags command to create or modify a tag. For example, the following command adds the **`Name=my-asg`** and **`costcenter=cc123`** tags. The tags are also added to any instances launched in the Auto Scaling group after this change. If a tag with either key already exists, the existing tag is replaced. The Amazon EC2 console associates the display name for each instance with the name that is specified for the `Name` key (case-sensitive).

```
aws autoscaling create-or-update-tags \
  --tags ResourceId=my-asg,ResourceType=auto-scaling-group,Key=Name,Value=my-
asg,PropagateAtLaunch=true \
  ResourceId=my-asg,ResourceType=auto-scaling-
group,Key=costcenter,Value=cc123,PropagateAtLaunch=true
```

Describe the tags for an Auto Scaling group (AWS CLI)

If you want to view the tags that are applied to a specific Auto Scaling group, you can use either of the following commands:

- describe-tags – You supply your Auto Scaling group name to view a list of the tags for the specified group.

  ```
  aws autoscaling describe-tags --filters Name=auto-scaling-group,Values=my-asg
  ```

  The following is an example response.

  ```
  {
      "Tags": [
          {
              "ResourceType": "auto-scaling-group",
              "ResourceId": "my-asg",
              "PropagateAtLaunch": true,
              "Value": "production",
              "Key": "environment"
          }
      ]
  ```

```
}
```

- describe-auto-scaling-groups – You supply your Auto Scaling group name to view the attributes of the specified group, including any tags.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

The following is an example response.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg",
            "AutoScalingGroupARN": "arn",
            "LaunchTemplate": {
                "LaunchTemplateId": "lt-0b97f1e282EXAMPLE",
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Latest"
            },
            "MinSize": 1,
            "MaxSize": 5,
            "DesiredCapacity": 1,
            ...
            "Tags": [
                {
                    "ResourceType": "auto-scaling-group",
                    "ResourceId": "my-asg",
                    "PropagateAtLaunch": true,
                    "Value": "production",
                    "Key": "environment"
                }
            ],
            ...
        }
    ]
}
```

Delete tags

You can delete a tag associated with your Auto Scaling group at any time.

Contents

- Delete tags (console)
- Delete tags (AWS CLI)

Delete tags (console)

To delete a tag

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Tags**, **Edit**.
4. Choose **Remove** next to the tag.
5. Choose **Update**.

Delete tags (AWS CLI)

Use the delete-tags command to delete a tag. For example, the following command deletes a tag with a key of **`environment`**.

```
aws autoscaling delete-tags --tags "ResourceId=my-asg,ResourceType=auto-scaling-
group,Key=environment"
```

You must specify the tag key, but you don't have to specify the value. If you specify a value and the value is incorrect, the tag is not deleted.

Tags for security

Use tags to verify that the requester (such as an IAM user or role) has permissions to create, modify, or delete specific Auto Scaling groups. Provide tag information in the condition element of an IAM policy by using one or more of the following condition keys:

- Use `autoscaling:ResourceTag/`*`tag-key`*`:` *`tag-value`* to allow (or deny) user actions on Auto Scaling groups with specific tags.

- Use `aws:RequestTag/tag-key: tag-value` to require that a specific tag be present (or not present) in a request.
- Use `aws:TagKeys [tag-key, ...]` to require that specific tag keys be present (or not present) in a request.

For example, you could deny access to all Auto Scaling groups that include a tag with the key **`environment`** and the value **`production`**, as shown in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:DeleteAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"autoscaling:ResourceTag/environment": "production"}
            }
        }
    ]
}
```

For more information about using condition keys to control access to Auto Scaling groups, see How Amazon EC2 Auto Scaling works with IAM.

Control access to tags

Use tags to verify that the requester (such as an IAM user or role) has permissions to add, modify, or delete tags for Auto Scaling groups.

The following example IAM policy gives the principal permission to remove only the tag with the **`temporary`** key from Auto Scaling groups.

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

```
            "Effect": "Allow",
            "Action": "autoscaling:DeleteTags",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": { "aws:TagKeys": ["temporary"] }
            }
        }
    ]
}
```

For more examples of IAM policies that enforce constraints on the tags specified for Auto Scaling groups, see Control which tag keys and tag values can be used.

> **Note**
>
> Even if you have a policy that restricts your users from performing a tagging (or untagging) operation on an Auto Scaling group, this does not prevent them from manually changing the tags on the instances after they have launched. For examples that control access to tags on EC2 instances, see Example: Tagging resources in the *Amazon EC2 User Guide*.

Use tags to filter Auto Scaling groups

The following examples show you how to use filters with the describe-auto-scaling-groups command to describe Auto Scaling groups with specific tags. Filtering by tags is limited to the AWS CLI or an SDK, and is not available from the console.

Filtering considerations

- You can specify multiple filters and multiple filter values in a single request.
- You cannot use wildcards with the filter values.
- Filter values are case-sensitive.

Example: Describe Auto Scaling groups with a specific tag key and value pair

The following command shows how to filter results to show only Auto Scaling groups with the tag key and value pair of **`environment=production`**.

```
aws autoscaling describe-auto-scaling-groups \
```

```
  --filters Name=tag-key,Values=environment Name=tag-value,Values=production
```

The following is an example response.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg",
            "AutoScalingGroupARN": "arn",
            "LaunchTemplate": {
                "LaunchTemplateId": "lt-0b97f1e282EXAMPLE",
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Latest"
            },
            "MinSize": 1,
            "MaxSize": 5,
            "DesiredCapacity": 1,
            ...
            "Tags": [
                {
                    "ResourceType": "auto-scaling-group",
                    "ResourceId": "my-asg",
                    "PropagateAtLaunch": true,
                    "Value": "production",
                    "Key": "environment"
                }
            ],
            ...
        },

    ... additional groups ...

    ]
}
```

Alternatively, you can specify tags using a `tag:<key>` filter. For example, the following command shows how to filter results to show only Auto Scaling groups with a tag key and value pair of **`environment=production`**. This filter is formatted as follows: `Name=tag:<key>,Values=<value>`, with **`<key>`** and **`<value>`** representing a tag key and value pair.

```
aws autoscaling describe-auto-scaling-groups \
```

```
  --filters Name=tag:environment,Values=production
```

You can also filter AWS CLI output by using the `--query` option. The following example shows how to limit AWS CLI output for the previous command to the group name, minimum size, maximum size, and desired capacity attributes only.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag:environment,Values=production \
  --query "AutoScalingGroups[].{AutoScalingGroupName: AutoScalingGroupName, MinSize:
 MinSize, MaxSize: MaxSize, DesiredCapacity: DesiredCapacity}"
```

The following is an example response.

```
[
    {
        "AutoScalingGroupName": "my-asg",
        "MinSize": 0,
        "MaxSize": 10,
        "DesiredCapacity": 1
    },

    ... additional groups ...

]
```

For more information about filtering, see Filtering AWS CLI output in the *AWS Command Line Interface User Guide*.

Example: Describe Auto Scaling groups with tags that match the tag key specified

The following command shows how to filter results to show only Auto Scaling groups with the **`environment`** tag, regardless of the tag value.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag-key,Values=environment
```

Example: Describe Auto Scaling groups with tags that match the set of tag keys specified

The following command shows how to filter results to show only Auto Scaling groups with tags for **`environment`** and **`project`**, regardless of the tag values.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag-key,Values=environment Name=tag-key,Values=project
```

Example: Describe Auto Scaling groups with tags that match at least one of the tag keys specified

The following command shows how to filter results to show only Auto Scaling groups with tags for **`environment`** or **`project`**, regardless of the tag values.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag-key,Values=environment,project
```

Example: Describe Auto Scaling groups with the specified tag value

The following command shows how to filter results to show only Auto Scaling groups with a tag value of **`production`**, regardless of the tag key.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag-value,Values=production
```

Example: Describe Auto Scaling groups with the set of tag values specified

The following command shows how to filter results to show only Auto Scaling groups with the tag values **`production`** and **`development`**, regardless of the tag key.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag-value,Values=production Name=tag-value,Values=development
```

Example: Describe Auto Scaling groups with tags that match at least one of the tag values specified

The following command shows how to filter results to show only Auto Scaling groups with a tag value of **`production`** or **`development`**, regardless of the tag key.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag-value,Values=production,development
```

Example: Describe Auto Scaling groups with tags that match multiple tag keys and values

You can also combine filters to create custom AND and OR logic to do more complex filtering.

The following command shows how to filter results to show only Auto Scaling groups with a specific set of tags. One tag key is **environment** AND the tag value is (**production** OR **development**) AND the other tag key is **costcenter** AND the tag value is **cc123**.

```
aws autoscaling describe-auto-scaling-groups \
  --filters Name=tag:environment,Values=production,development
 Name=tag:costcenter,Values=cc123
```

Instance maintenance policies

You can configure an instance maintenance policy for your Auto Scaling group to meet specific capacity requirements during events that cause instances to be replaced, such as an instance refresh or the health check process.

For example, suppose you have an Auto Scaling group that has a small number of instances. You want to avoid the potential disruptions from terminating and then replacing an instance when health checks indicate an impaired instance. With an instance maintenance policy, you can make sure that Amazon EC2 Auto Scaling first launches a new instance and then waits for it to be fully ready before terminating the unhealthy instance.

An instance maintenance policy also helps you minimize any potential disruptions in cases where multiple instances are replaced at the same time. You set the minimum and maximum healthy percentage parameters for the policy, and your Auto Scaling group can only increase and decrease capacity within that minimum-maximum range when replacing instances. A larger range increases the number of instances that can be replaced at the same time.

Contents

- Instance maintenance policy for Auto Scaling group
- Set an instance maintenance policy on your Auto Scaling group

Instance maintenance policy for Auto Scaling group

This topic provides an overview of the options available and describes what to consider when you create an instance maintenance policy.

Contents

- Overview

- Core concepts
- Instance warmup
- Health check grace period
- Scale your Auto Scaling group
- Example scenarios

Overview

When you create an instance maintenance policy for your Auto Scaling group, the policy affects Amazon EC2 Auto Scaling events that cause instances to be replaced. This results in more consistent replacement behaviors within the same Auto Scaling group. It also lets you optimize your group for availability or cost depending on your needs.

In the console, the following configuration options are available:

- **Launch before terminating** – A new instance must be provisioned first before an existing instance can be terminated. This approach is a good choice for applications that favor availability over cost savings.
- **Terminate and launch** – New instances are provisioned at the same time your existing instances are terminated. This approach is a good choice for applications that favor cost savings over availability. It's also a good choice for applications that should not launch more capacity than is currently available, even when replacing instances.
- **Custom policy** – This option lets you set up your policy with a custom minimum and maximum range for the amount of capacity that you want available when replacing instances. This approach can help you achieve the right balance between cost and availability.

The default for an Auto Scaling group is to not have an instance maintenance policy, which causes it to respond to instance maintenance events with the default behaviors. The default behaviors are described in the following table.

Instance maintenance event default behaviors

Event	Description	Default behavior
Health check failure	Happens automatically when instances fail their health checks. Amazon EC2 Auto	Terminate and launch.

Event	Description	Default behavior
	Scaling replaces instances that fail their health checks. To understand the causes of health check failures, see Health checks for instances in an Auto Scaling group.	
Instance refresh	Happens when you start an instance refresh. Depending on your configuration, an instance refresh replaces instances one at a time, several at a time, or all at once. For more information, see Use an instance refresh to update instances in an Auto Scaling group.	Terminate and launch.
Maximum instance lifetime	Happens automatically when instances reach the maximum instance lifetime that you specify for your Auto Scaling group. Amazon EC2 Auto Scaling replaces instances that reach their maximum instance lifetime. For more information, see Replace Auto Scaling instances based on maximum instance lifetime.	Terminate and launch.

Event	Description	Default behavior
Rebalancing	Happens automatically if there are underlying changes that cause the group to become unbalanced. Amazon EC2 Auto Scaling rebalance s the group in the following situations: • An Availability Zone that previously had insufficient capacity recovers, or you add or remove an Availabil ity Zone from the group. When this happens, your Auto Scaling group tries to evenly balance itself across Availability Zones. For more information, see Rebalanci ng activities. • You enable Capacity Rebalancing on your Auto Scaling group, and it tries to launch new Spot Instances before existing ones are interrupted as the availability of Spot Instances changes. For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions. • You update your Auto Scaling group, and it gradually replaces instances	Launch before terminating. Amazon EC2 Auto Scaling can exceed your group's size limits by up to 10 percent of its *maximum capacity*. However, if you're using Capacity Rebalancing, it can only exceed these limits by up to 10 percent of the *desired capacity*.

Event	Description	Default behavior
	to match the new purchasin g options that you chose when updating a mixed instances policy. For more information, see Update an Auto Scaling group.	

Amazon EC2 Auto Scaling will continue to default to terminate and launch in the following situations. Therefore, when one of these situations occur, your group's capacity might be less than the lower threshold of your instance maintenance policy.

- When an instance terminates unexpectedly, for example, because of human action. Amazon EC2 Auto Scaling immediately replaces instances that are no longer running. For more information, see Amazon EC2 health checks.
- When Amazon EC2 reboots, stops, or retires an instance as part of a scheduled event before Amazon EC2 Auto Scaling can launch the replacement instance. For more information about these events, see Scheduled events for your instances in the *Amazon EC2 User Guide*.
- When the Amazon EC2 Spot Service initiates a Spot Instance interruption and a Spot Instance is then forcibly terminated.

With Spot Instances, if you enabled Capacity Rebalancing on your Auto Scaling group, then the instance might already have a pending instance from a different Spot pool that we launched before we initiated the Spot interruption. For details about how Capacity Rebalancing works, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

However, because Spot Instances are not guaranteed to remain available and can be terminated with a two-minute Spot Instance interruption notice, your instance maintenance policy's lower threshold can be exceeded if instances are interrupted before your new instances have launched.

Core concepts

Before you get started, familiarize yourself with the following core concepts and terms:

Desired capacity

The *desired capacity* is the capacity of the Auto Scaling group at the time of creation. It is also the capacity the group attempts to maintain when there are no scaling conditions attached to the group.

Instance maintenance policy

An *instance maintenance policy* controls whether an instance is provisioned first before an existing instance is terminated for instance maintenance events. It also determines how far below and over your desired capacity your Auto Scaling group might go to replace multiple instances at the same time.

Maximum healthy percentage

The *maximum healthy percentage* is the percentage of its desired capacity that your Auto Scaling group can increase to when replacing instances. It represents the maximum percentage of the group that can be in service and healthy, or pending, to support your workload. In the console, you can set the maximum healthy percentage when you use either the **Launch before terminating** option or the **Custom policy** option. The valid values are 100–200 percent.

Minimum healthy percentage

The *minimum healthy percentage* is the percentage of the desired capacity to keep in service, healthy, and ready to use to support your workload when replacing instances. An instance is considered healthy and ready to use after it successfully completes its first health check and the specified warmup time passes. In the console, you can set the minimum healthy percentage when you use either the **Terminate and launch** option or the **Custom policy** option. The valid values are 0–100 percent.

> **Note**
>
> To replace instances faster, you can specify a low minimum healthy percentage. However, if there aren't enough healthy instances running, it can reduce availability. We recommend selecting a reasonable value to maintain availability in situations where multiple instances will be replaced.

Instance warmup

If your instances need time to initialize after they enter the `InService` state, enable the default instance warmup for your Auto Scaling group. With the default instance warmup, you can prevent instances from being counted toward the minimum healthy percentage before they are ready. This ensures that Amazon EC2 Auto Scaling considers how long it takes to have enough capacity in place to support the workload before it terminates existing instances.

As an added benefit, you can improve the Amazon CloudWatch metrics used for dynamic scaling when you enable the default instance warmup. If your Auto Scaling group has any scaling policies, when the group scales out, it uses the same default warmup period to prevent instances from being counted toward CloudWatch metrics before they have finished initializing.

For more information, see Set the default instance warmup for an Auto Scaling group.

Health check grace period

Amazon EC2 Auto Scaling determines whether an instance is healthy based on the status of the health checks that your Auto Scaling group uses. For more information, see Health checks for instances in an Auto Scaling group.

To make sure that these health checks start as soon as possible, don't set the group's health check grace period too high, but high enough for your Elastic Load Balancing health checks to determine whether a target is available to handle requests. For more information, see Set the health check grace period for an Auto Scaling group.

Scale your Auto Scaling group

An instance maintenance policy only applies to instance maintenance events and doesn't prevent the group from being manually or automatically scaled.

When there are scaling policies or scheduled actions attached to your Auto Scaling group, they can run in parallel while instance maintenance events are occurring. In which case, they could increase or decrease the group's desired capacity but only within the scaling limits that you defined. For more information about these limits, see Set scaling limits for your Auto Scaling group.

Example scenarios

In a typical scenario, your instance maintenance policy and desired capacity might look something like this:

- Minimum healthy percentage = 90 percent
- Maximum healthy percentage = 120 percent
- Desired capacity = 100

During any instance maintenance event, your Auto Scaling group might have as few as 90 instances and as many as 120. After the event, the group goes back to having 100 instances.

When you use an instance maintenance policy with an Auto Scaling group that has a warm pool, the minimum and maximum healthy percentages are applied separately to the Auto Scaling group and the warm pool.

For example, assume this is your configuration:

- Minimum healthy percentage = 90 percent
- Maximum healthy percentage = 120 percent
- Desired capacity = 100
- Warm pool size = 10

If you start an instance refresh to recycle the group's instances, Amazon EC2 Auto Scaling replaces instances in the Auto Scaling group first, and then instances in the warm pool. While Amazon EC2 Auto Scaling is still working on replacing instances in the Auto Scaling group, the group might have as few as 90 instances and as many as 120. After finishing with the group, Amazon EC2 Auto Scaling can work on replacing instances in the warm pool. While this is happening, the warm pool might have as few as 9 instances and as many as 12.

Set an instance maintenance policy on your Auto Scaling group

You can create an instance maintenance policy when you create an Auto Scaling group. You can also create it for existing groups.

By setting an instance maintenance policy on your Auto Scaling group, you no longer have to specify values for minimum and maximum healthy percentage parameters for the instance refresh feature unless you want to override the instance maintenance policy.

In the console, Amazon EC2 Auto Scaling provides options to help you get started.

Contents

- Set an instance maintenance policy

- Remove an instance maintenance policy

Set an instance maintenance policy

To set an instance maintenance policy on an Auto Scaling group, use one of the following methods:

Console

To set an instance maintenance policy on a new group (console)

1. Follow the instructions in Create an Auto Scaling group using a launch template and complete each step in the procedure, up to step 11.
2. On the **Configure group size and scaling policies**, for **Desired capacity**, enter the initial number of instances to launch.
3. In the **Scaling** section, under **Scaling limits**, if your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed.
4. For **Automatic scaling**, choose whether you want to create a target tracking scaling policy. You can also create this policy after your create your Auto Scaling group.

   If you choose **Target tracking scaling policy**, follow the directions in Create a target tracking scaling policy to create the policy.
5. In the **Instance maintenance policy** section, choose one of the available options:

   - **Launch before terminating**: A new instance must be provisioned first before an existing instance can be terminated. This is a good choice for applications that favor availability over cost savings.
   - **Terminate and launch**: New instances are provisioned at the same time your existing instances are terminated. This is a good choice for applications that favor cost savings over availability. It's also a good choice for applications that should not launch more capacity than is currently available.
   - **Custom policy**: This option lets you set up your policy with a custom minimum and maximum range for the amount of capacity that you want available when replacing instances. This can help you achieve the right balance between cost and availability.
6. For **Set healthy percentage**, enter values for one or both of the following fields. The enabled fields vary depending on the option that you chose in the preceding step.

- **Min**: Sets the minimum healthy percentage that's required to proceed with replacing instances.
- **Max**: Sets the maximum healthy percentage that's possible when replacing instances.

7. Expand the **View capacity during replacements based on your desired capacity** section to confirm how the values for **Min** and **Max** apply to your group. The exact values used depend on the desired capacity value, which will change if the group scales.
8. Continue with the steps in Create an Auto Scaling group using a launch template.

AWS CLI

To set an instance maintenance policy on a new group (AWS CLI)

Add the `--instance-maintenance-policy` option to the create-auto-scaling-group command. The following example set an instance maintenance policy on a new Auto Scaling group named *my-asg*.

```
aws autoscaling create-auto-scaling-group \
  --launch-template LaunchTemplateName=my-launch-template,Version='1' \
  --auto-scaling-group-name my-asg \
  --min-size 1 \
  --max-size 10 \
  --desired-capacity 5 \
  --default-instance-warmup 20 \
  --instance-maintenance-policy '{
      "MinHealthyPercentage": 90,
      "MaxHealthyPercentage": 120
    }' \
  --vpc-zone-identifier "subnet-5e6example,subnet-613example,subnet-c93example"
```

Console

To set an instance maintenance policy on an existing group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your Auto Scaling group in.

3. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
4. On the **Details** tab, choose **Instance maintenance policy**, **Edit**.
5. To set an instance maintenance policy on the group, choose one of the available options:

   - **Launch before terminating**: A new instance must be provisioned first before an existing instance can be terminated. This is a good choice for applications that favor availability over cost savings.
   - **Terminate and launch**: New instances are provisioned at the same time your existing instances are terminated. This is a good choice for applications that favor cost savings over availability. It's also a good choice for applications that should not launch more capacity than is currently available.
   - **Custom policy**: This option lets you set up your policy with a custom minimum and maximum range for the amount of capacity that you want available when replacing instances. This can help you achieve the right balance between cost and availability.
6. For **Set healthy percentage**, enter values for one or both of the following fields. The enabled fields vary depending on the option that you chose in the preceding step.

   - **Min**: Sets the minimum healthy percentage that's required to proceed with replacing instances.
   - **Max**: Sets the maximum healthy percentage that's possible when replacing instances.
7. Expand the **View capacity during replacements based on your desired capacity** section to confirm how the values for **Min** and **Max** apply to your group. The exact values used depend on the desired capacity value, which will change if the group scales.
8. Choose **Update**.

AWS CLI

To set an instance maintenance policy on an existing group (AWS CLI)

Add the `--instance-maintenance-policy` option to the update-auto-scaling-group command. The following example sets an instance maintenance policy on the specified Auto Scaling group.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --instance-maintenance-policy '{
```

```
      "MinHealthyPercentage": 90,
      "MaxHealthyPercentage": 120
    }'
```

Remove an instance maintenance policy

If you want to stop using an instance maintenance policy with your Auto Scaling group, you can remove it.

Console

To remove an instance maintenance policy (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your Auto Scaling group in.
3. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
4. On the **Details** tab, choose **Instance maintenance policy**, **Edit**.
5. Choose **No instance maintenance policy**.
6. Choose **Update**.

AWS CLI

To remove an instance maintenance policy (AWS CLI)

Add the `--instance-maintenance-policy` option to the update-auto-scaling-group command. The following example removes the instance maintenance policy from the specified Auto Scaling group.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --instance-maintenance-policy '{
      "MinHealthyPercentage": -1,
      "MaxHealthyPercentage": -1
    }'
```

Amazon EC2 Auto Scaling lifecycle hooks

Amazon EC2 Auto Scaling offers the ability to add lifecycle hooks to your Auto Scaling groups. These hooks let you create solutions that are aware of events in the Auto Scaling instance lifecycle, and then perform a custom action on instances when the corresponding lifecycle event occurs. A lifecycle hook provides a specified amount of time (one hour by default) to wait for the action to complete before the instance transitions to the next state.

As an example of using lifecycle hooks with Auto Scaling instances:

- When a scale-out event occurs, your newly launched instance completes its startup sequence and transitions to a wait state. While the instance is in a wait state, it runs a script to download and install the needed software packages for your application, making sure that your instance is fully ready before it starts receiving traffic. When the script is finished installing software, it sends the **complete-lifecycle-action** command to continue.
- When a scale-in event occurs, a lifecycle hook pauses the instance before it is terminated and sends you a notification using Amazon EventBridge. While the instance is in the wait state, you can invoke an AWS Lambda function or connect to the instance to download logs or other data before the instance is fully terminated.

A popular use of lifecycle hooks is to control when instances are registered with Elastic Load Balancing. By adding a launch lifecycle hook to your Auto Scaling group, you can ensure that your bootstrap scripts have completed successfully and the applications on the instances are ready to accept traffic before they are registered to the load balancer at the end of the lifecycle hook.

Contents

- Lifecycle hook availability
- Considerations and limitations for lifecycle hooks
- Related resources
- How lifecycle hooks work in Auto Scaling groups
- Prepare to add a lifecycle hook to your Auto Scaling group
- Retrieve the target lifecycle state through instance metadata
- Add lifecycle hooks to your Auto Scaling group
- Complete a lifecycle action in an Auto Scaling group
- Tutorial: Use data script and instance metadata to retrieve lifecycle state

- Tutorial: Configure a lifecycle hook that invokes a Lambda function

Lifecycle hook availability

The following table lists the lifecycle hooks available for various scenarios.

Event	Instance launch or termination[1]	Maximum Instance Lifetime: Replacement instances	Instance Refresh: Replacement instances	Capacity Rebalancing: Replacement instances	Warm Pools: Instances entering and leaving the warm pool
Instance launching	✓	✓	✓	✓	✓
Instance terminating	✓	✓	✓	✓	✓

[1] Applies to all launches and terminations, whether they are initiated automatically or manually such as when you call the `SetDesiredCapacity` or `TerminateInstanceInAutoScalingGroup` operations. Does not apply when you attach or detach instances, move instances in and out of standby mode, or delete the group with the force delete option.

Considerations and limitations for lifecycle hooks

When working with lifecycle hooks, keep in mind the following notes and limitations:

- Amazon EC2 Auto Scaling provides its own lifecycle to help with the management of Auto Scaling groups. This lifecycle differs from that of other EC2 instances. For more information, see Amazon EC2 Auto Scaling instance lifecycle. Instances in a warm pool also have their own lifecycle, as described in Lifecycle state transitions for instances in a warm pool.
- You can use lifecycle hooks with Spot Instances, but a lifecycle hook does not prevent an instance from terminating in the event that capacity is no longer available, which can happen at any time with a two-minute interruption notice. For more information, see Spot Instance interruptions in the *Amazon EC2 User Guide*. However, you can enable Capacity Rebalancing

to proactively replace Spot Instances that have received a rebalance recommendation from the Amazon EC2 Spot service, a signal that is sent when a Spot Instance is at elevated risk of interruption. For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

- Instances can remain in a wait state for a finite period of time. The default timeout for a lifecycle hook is one hour (heartbeat timeout). There is also a global timeout that specifies the maximum amount of time that you can keep an instance in a wait state. The global timeout is 48 hours or 100 times the heartbeat timeout, whichever is smaller.
- The result of the lifecycle hook can be either abandon or continue. If an instance is launching, continue indicates that your actions were successful, and that Amazon EC2 Auto Scaling can put the instance into service. Otherwise, abandon indicates that your custom actions were unsuccessful, and that we can terminate and replace the instance. If an instance is terminating, both abandon and continue allow the instance to terminate. However, abandon stops any remaining actions, such as other lifecycle hooks, and continue allows any other lifecycle hooks to complete.
- Amazon EC2 Auto Scaling limits the rate at which it allows instances to launch if the lifecycle hooks are failing consistently, so make sure to test and fix any permanent errors in your lifecycle actions.
- Creating and updating lifecycle hooks using the AWS CLI, AWS CloudFormation, or an SDK provides options not available when creating a lifecycle hook from the AWS Management Console. For example, the field to specify the ARN of an SNS topic or SQS queue doesn't appear in the console, because Amazon EC2 Auto Scaling already sends events to Amazon EventBridge. These events can be filtered and redirected to AWS services such as Lambda, Amazon SNS, and Amazon SQS as needed.
- You can add multiple lifecycle hooks to an Auto Scaling group while you are creating it, by calling the CreateAutoScalingGroup API using the AWS CLI, AWS CloudFormation, or an SDK. However, each hook must have the same notification target and IAM role, if specified. To create lifecycle hooks with different notification targets and different roles, create the lifecycle hooks one at a time in separate calls to the PutLifecycleHook API.
- If you add a lifecycle hook for instance launch, the health check grace period starts as soon as the instance reaches the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.

Scaling considerations

- Dynamic scaling policies scale in and out in response to CloudWatch metric data, such as CPU and network I/O, that's aggregated across multiple instances. When scaling out, Amazon EC2 Auto Scaling doesn't immediately count a new instance towards the aggregated instance metrics of the Auto Scaling group. It waits until the instance reaches the `InService` state and the instance warmup has finished. For more information, see Scaling performance considerations in the default instance warmup topic.
- On scale in, the aggregated instance metrics might not instantly reflect the removal of a terminating instance. The terminating instance stops counting toward the group's aggregated instance metrics shortly after the Amazon EC2 Auto Scaling termination workflow begins.
- In most cases when lifecycle hooks are invoked, scaling activities due to simple scaling policies are paused until the lifecycle actions have completed and the cooldown period has expired. Setting a long interval for the cooldown period means that it will take longer for scaling to resume. For more information, see Lifecycle hooks can cause additional delays in the cooldown topic. In general, we recommend against using simple scaling policies if you can use either step scaling or target tracking scaling policies instead.

Related resources

For an introduction video, see AWS re:Invent 2018: Capacity Management Made Easy with Amazon EC2 Auto Scaling on *YouTube*.

We provide a few JSON and YAML template snippets that you can use to understand how to declare lifecycle hooks in your AWS CloudFormation stack templates. For more information, see the AWS::AutoScaling::LifecycleHook reference in the *AWS CloudFormation User Guide*.

You can also visit our GitHub repository to download example templates and user data scripts for lifecycle hooks.

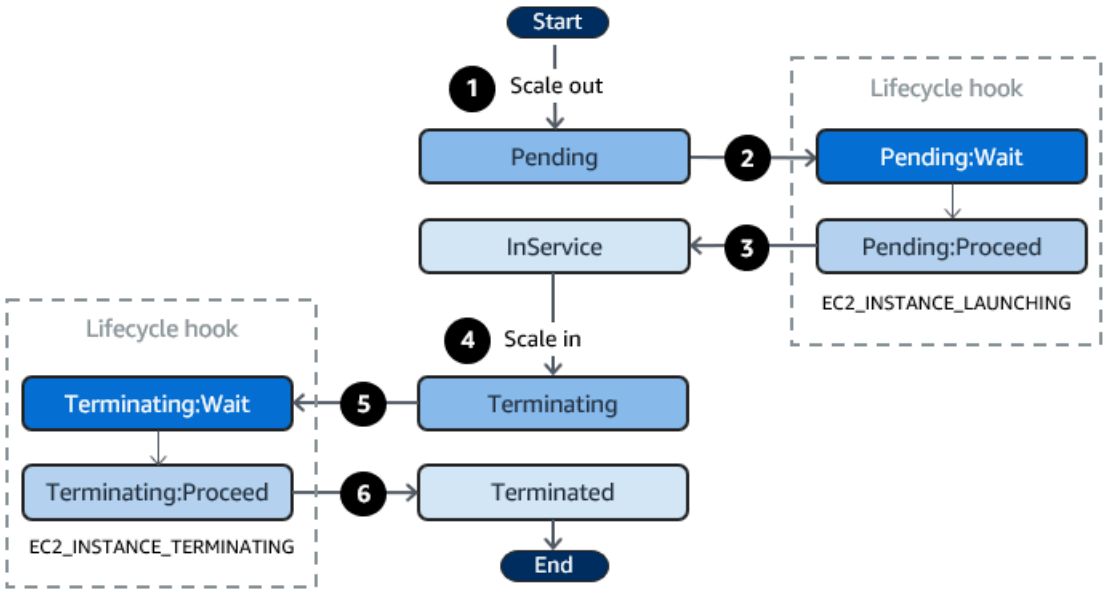
For examples of the use of lifecycle hooks, see the following blog posts.

- Building a Backup System for Scaled Instances using Lambda and Amazon EC2 Run Command
- Run code before terminating an EC2 Auto Scaling instance.

How lifecycle hooks work in Auto Scaling groups

An Amazon EC2 instance transitions through different states from the time it launches until it is terminated. You can create custom actions for your Auto Scaling group to act when an instance transitions into a wait state due to a lifecycle hook.

The following illustration shows the transitions between Auto Scaling instance states when you use lifecycle hooks for scale out and scale in.



As shown in the preceding diagram:

1. The Auto Scaling group responds to a scale-out event and begins launching an instance.
2. The lifecycle hook puts the instance into a wait state (`Pending:Wait`) and then performs a custom action.

   The instance remains in a wait state until you either complete the lifecycle action, or the timeout period ends. By default, the instance remains in a wait state for one hour, and then the Auto Scaling group continues the launch process (`Pending:Proceed`). If you need more time, you can restart the timeout period by recording a heartbeat. If you complete the lifecycle action when the custom action has completed and the timeout period hasn't expired yet, the period ends and the Auto Scaling group continues the launch process.
3. The instance enters the `InService` state and the health check grace period starts. However, before the instance reaches the `InService` state, if the Auto Scaling group is associated with an Elastic Load Balancing load balancer, the instance is registered with the load balancer, and the

load balancer starts checking its health. After the health check grace period ends, Amazon EC2 Auto Scaling begins checking the health state of the instance.

4. The Auto Scaling group responds to a scale-in event and begins terminating an instance. If the Auto Scaling group is being used with Elastic Load Balancing, the terminating instance is first deregistered from the load balancer. If connection draining is enabled for the load balancer, the instance stops accepting new connections and waits for existing connections to drain before completing the deregistration process.
5. The lifecycle hook puts the instance into a wait state (`Terminating:Wait`) and then performs a custom action.

   The instance remains in a wait state either until you complete the lifecycle action, or until the timeout period ends (one hour by default). After you complete the lifecycle hook or the timeout period expires, the instance transitions to the next state (`Terminating:Proceed`).
6. The instance is terminated.

> **Important**
>
> Instances in a warm pool also have their own lifecycle with corresponding wait states, as described in Lifecycle state transitions for instances in a warm pool.

Prepare to add a lifecycle hook to your Auto Scaling group

Before you add a lifecycle hook to your Auto Scaling group, be sure that your user data script or notification target is set up correctly.

- To use a user data script to perform custom actions on your instances as they are launching, you do not need to configure a notification target. However, you must have already created the launch template or launch configuration that specifies your user data script and associated it with your Auto Scaling group. For more information about user data scripts, see Run commands on your Linux instance at launch in the *Amazon EC2 User Guide*.
- To signal Amazon EC2 Auto Scaling when the lifecycle action is complete, you must add the CompleteLifecycleAction API call to the script, and you must manually create an IAM role with a policy that allows Auto Scaling instances to call this API. Your launch template or launch configuration must specify this role using an IAM instance profile that gets attached to your

Amazon EC2 instances at launch. For more information, see Complete a lifecycle action in an Auto Scaling group and IAM role for applications that run on Amazon EC2 instances.

- To use a service such as Lambda to perform a custom action, you must have already created an EventBridge rule and specified a Lambda function as its target. For more information, see Configure a notification target for lifecycle notifications.
- To allow Lambda to signal Amazon EC2 Auto Scaling when the lifecycle action is complete, you must add the CompleteLifecycleAction API call to the function code. You must also have attached an IAM policy to the function's execution role that gives Lambda permission to complete lifecycle actions. For more information, see Tutorial: Configure a lifecycle hook that invokes a Lambda function.
- To use a service such as a Amazon SNS or Amazon SQS to perform a custom action, you must have already created the SNS topic or SQS queue and have ready its Amazon Resource Name (ARN). You must also have already created the IAM role that gives Amazon EC2 Auto Scaling access to your SNS topic or SQS target and have ready its ARN. For more information, see Configure a notification target for lifecycle notifications.

> **Note**
>
> By default, when you add a lifecycle hook in the console, Amazon EC2 Auto Scaling sends lifecycle event notifications to Amazon EventBridge. Using EventBridge or a user data script is a recommended best practice. To create a lifecycle hook that sends notifications directly to Amazon SNS or Amazon SQS, use the AWS CLI, AWS CloudFormation, or an SDK to add the lifecycle hook.

Configure a notification target for lifecycle notifications

You can add lifecycle hooks to an Auto Scaling group to perform custom actions when an instance enters a wait state. You can choose a target service to perform these actions depending on your preferred development approach.

The first approach uses Amazon EventBridge to invoke a Lambda function that performs the action you want. The second approach involves creating an Amazon Simple Notification Service (Amazon SNS) topic to which notifications are published. Clients can subscribe to the SNS topic and receive published messages using a supported protocol. The last approach involves using Amazon Simple Queue Service (Amazon SQS), a messaging system used by distributed applications to exchange messages through a polling model.

As a best practice, we recommend that you use EventBridge. The notifications sent to Amazon SNS and Amazon SQS contain the same information as the notifications that Amazon EC2 Auto Scaling sends to EventBridge. Before EventBridge, the standard practice was to send a notification to SNS or SQS and integrate another service with SNS or SQS to perform programmatic actions. Today, EventBridge gives you more options for which services you can target and makes it easier to handle events using serverless architecture.

The following procedures cover how to set up your notification target.

Remember, if you have a user data script in your launch template or launch configuration that configures your instances when they launch, you do not need to receive notifications to perform custom actions on your instances.

Contents

- Route notifications to Lambda using EventBridge
- Receive notifications using Amazon SNS
- Receive notifications using Amazon SQS
- Notification message example for Amazon SNS and Amazon SQS

> **Important**
>
> The EventBridge rule, Lambda function, Amazon SNS topic, and Amazon SQS queue that you use with lifecycle hooks must always be in the same Region where you created your Auto Scaling group.

Route notifications to Lambda using EventBridge

You can configure an EventBridge rule to invoke a Lambda function when an instance enters a wait state. Amazon EC2 Auto Scaling emits a lifecycle event notification to EventBridge about the instance that is launching or terminating and a token that you can use to control the lifecycle action. For examples of these events, see Amazon EC2 Auto Scaling event reference.

> **Note**
>
> When you use the AWS Management Console to create an event rule, the console automatically adds the IAM permissions necessary to grant EventBridge permission to call

your Lambda function. If you are creating an event rule using the AWS CLI, you need to grant this permission explicitly.
For information about how to create event rules in the EventBridge console, see Creating Amazon EventBridge rules that react to events in the *Amazon EventBridge User Guide*.
– or –
For an introductory tutorial that is directed towards console users, see Tutorial: Configure a lifecycle hook that invokes a Lambda function. This tutorial shows you how to create a simple Lambda function that listens for launch events and writes them out to a CloudWatch Logs log.

To create an EventBridge rule that invokes a Lambda function

1. Create a Lambda function by using the Lambda console and note its Amazon Resource Name (ARN). For example, `arn:aws:lambda:`*`region`*`:`*`123456789012`*`:function:`*`my-function`*. You need the ARN to create an EventBridge target. For more information, see Getting started with Lambda in the *AWS Lambda Developer Guide*.
2. To create a rule that matches events for instance launch, use the following put-rule command.

```
aws events put-rule --name my-rule --event-pattern file://pattern.json --state
 ENABLED
```

The following example shows the `pattern.json` for an instance launch lifecycle action. Replace the text in *`italics`* with the name of your Auto Scaling group.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-launch Lifecycle Action" ],
  "detail": {
      "AutoScalingGroupName": [ "my-asg" ]
   }
}
```

If the command runs successfully, EventBridge responds with the ARN of the rule. Note this ARN. You'll need to enter it in step 4.

To create a rule that matches for other events, modify the event pattern. For more information, see Use EventBridge to handle Auto Scaling events.

3. To specify the Lambda function to use as a target for the rule, use the following put-targets command.

```
aws events put-targets --rule my-rule --targets
 Id=1,Arn=arn:aws:lambda:region:123456789012:function:my-function
```

In the preceding command, *my-rule* is the name that you specified for the rule in step 2, and the value for the `Arn` parameter is the ARN of the function that you created in step 1.

4. To add permissions that allow the rule to invoke your Lambda function, use the following Lambda add-permission command. This command trusts the EventBridge service principal (`events.amazonaws.com`) and scopes permissions to the specified rule.

```
aws lambda add-permission --function-name my-function --statement-id my-unique-id \
  --action 'lambda:InvokeFunction' --principal events.amazonaws.com --source-arn
 arn:aws:events:region:123456789012:rule/my-rule
```

In the preceding command:

- *my-function* is the name of the Lambda function that you want the rule to use as a target.
- *my-unique-id* is a unique identifier that you define to describe the statement in the Lambda function policy.
- `source-arn` is the ARN of the EventBridge rule.

If the command runs successfully, you receive output similar to the following.

```
{
  "Statement": "{\"Sid\":\"my-unique-id\",
    \"Effect\":\"Allow\",
    \"Principal\":{\"Service\":\"events.amazonaws.com\"},
    \"Action\":\"lambda:InvokeFunction\",
    \"Resource\":\"arn:aws:lambda:us-west-2:123456789012:function:my-function\",
    \"Condition\":
      {\"ArnLike\":
        {\"AWS:SourceArn\":
         \"arn:aws:events:us-west-2:123456789012:rule/my-rule\"}}}"
}
```

The `Statement` value is a JSON string version of the statement that was added to the Lambda function policy.

5. After you have followed these instructions, continue on to Add lifecycle hooks to your Auto Scaling group as a next step.

Receive notifications using Amazon SNS

You can use Amazon SNS to set up a notification target (an SNS topic) to receive notifications when a lifecycle action occurs. Amazon SNS then sends the notifications to the subscribed recipients. Until the subscription is confirmed, no notifications published to the topic are sent to the recipients.

To set up notifications using Amazon SNS

1. Create an Amazon SNS topic by using either the Amazon SNS console or the following create-topic command. Ensure that the topic is in the same Region as the Auto Scaling group that you're using. For more information, see Getting started with Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.

```
aws sns create-topic --name my-sns-topic
```

2. Note the topic Amazon Resource Name (ARN), for example, `arn:aws:sns:region:123456789012:my-sns-topic`. You need it to create the lifecycle hook.
3. Create an IAM service role to give Amazon EC2 Auto Scaling access to your Amazon SNS notification target.

   To give Amazon EC2 Auto Scaling access to your SNS topic

   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the navigation pane on the left, choose **Roles**.
   c. Choose **Create role**.
   d. For **Select trusted entity**, choose **AWS service**.
   e. For your use case, under **Use cases for other AWS services**, choose **EC2 Auto Scaling** and then **EC2 Auto Scaling Notification Access**.
   f. Choose **Next** twice to go to the **Name, review, and create** page.

g. For **Role name**, enter a name for the role (for example, **`my-notification-role`**) and choose **Create role**.

h. On the **Roles** page, choose the role that you just created to open the **Summary** page. Make a note of the role **ARN**. For example, `arn:aws:iam::123456789012:role/my-notification-role`. You need it to create the lifecycle hook.

4. After you have followed these instructions, continue on to Add lifecycle hooks (AWS CLI) as a next step.

Receive notifications using Amazon SQS

You can use Amazon SQS to set up a notification target to receive messages when a lifecycle action occurs. A queue consumer must then poll an SQS queue to act on these notifications.

> **Important**
>
> FIFO queues are not compatible with lifecycle hooks.

To set up notifications using Amazon SQS

1. Create an Amazon SQS queue by using the Amazon SQS console. Ensure that the queue is in the same Region as the Auto Scaling group that you're using. For more information, see Getting started with Amazon SQS in the *Amazon Simple Queue Service Developer Guide*.
2. Note the queue ARN, for example, `arn:aws:sqs:us-west-2:123456789012:my-sqs-queue`. You need it to create the lifecycle hook.
3. Create an IAM service role to give Amazon EC2 Auto Scaling access to your Amazon SQS notification target.

   To give Amazon EC2 Auto Scaling access to your SQS queue

   a. Open the IAM console at https://console.aws.amazon.com/iam/.

   b. In the navigation pane on the left, choose **Roles**.

   c. Choose **Create role**.

   d. For **Select trusted entity**, choose **AWS service**.

   e. For your use case, under **Use cases for other AWS services**, choose **EC2 Auto Scaling** and then **EC2 Auto Scaling Notification Access**.

f. Choose **Next** twice to go to the **Name, review, and create** page.

g. For **Role name**, enter a name for the role (for example, **`my-notification-role`**) and choose **Create role**.

h. On the **Roles** page, choose the role that you just created to open the **Summary** page. Make a note of the role **ARN**. For example, `arn:aws:iam::`*`123456789012`*`:role/`*`my-notification-role`*. You need it to create the lifecycle hook.

4. After you have followed these instructions, continue on to Add lifecycle hooks (AWS CLI) as a next step.

Notification message example for Amazon SNS and Amazon SQS

While the instance is in a wait state, a message is published to the Amazon SNS or Amazon SQS notification target. The message includes the following information:

- `LifecycleActionToken` — The lifecycle action token.
- `AccountId` — The AWS account ID.
- `AutoScalingGroupName` — The name of the Auto Scaling group.
- `LifecycleHookName` — The name of the lifecycle hook.
- `EC2InstanceId` — The ID of the EC2 instance.
- `LifecycleTransition` — The lifecycle hook type.
- `NotificationMetadata` — The notification metadata.

The following is a notification message example.

```
Service: AWS Auto Scaling
Time: 2021-01-19T00:36:26.533Z
RequestId: 18b2ec17-3e9b-4c15-8024-ff2e8ce8786a
LifecycleActionToken: 71514b9d-6a40-4b26-8523-05e7ee35fa40
AccountId: 123456789012
AutoScalingGroupName: my-asg
LifecycleHookName: my-hook
EC2InstanceId: i-0598c7d356eba48d7
LifecycleTransition: autoscaling:EC2_INSTANCE_LAUNCHING
NotificationMetadata: hook message metadata
```

Test notification message example

When you first add a lifecycle hook, a test notification message is published to the notification target. The following is a test notification message example.

```
Service: AWS Auto Scaling
Time: 2021-01-19T00:35:52.359Z
RequestId: 18b2ec17-3e9b-4c15-8024-ff2e8ce8786a
Event: autoscaling:TEST_NOTIFICATION
AccountId: 123456789012
AutoScalingGroupName: my-asg
AutoScalingGroupARN: arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:042cba90-
ad2f-431c-9b4d-6d9055bcc9fb:autoScalingGroupName/my-asg
```

> **Note**
>
> For examples of the events delivered from Amazon EC2 Auto Scaling to EventBridge, see Amazon EC2 Auto Scaling event reference.

Retrieve the target lifecycle state through instance metadata

Each Auto Scaling instance that you launch goes through several lifecycle states. To invoke custom actions from within an instance that act on specific lifecycle state transitions, you must retrieve the target lifecycle state through instance metadata.

For example, you might need a mechanism to detect instance termination from inside the instance to run some code on the instance before it's terminated. You can do this by writing code that polls for the lifecycle state of an instance directly from the instance. You can then add a lifecycle hook to the Auto Scaling group to keep the instance running until your code sends the **complete-lifecycle-action** command to continue.

The Auto Scaling instance lifecycle has two primary steady states—`InService` and `Terminated`—and two side steady states—`Detached` and `Standby`. If you use a warm pool, the lifecycle has four additional steady states—`Warmed:Hibernated`, `Warmed:Running`, `Warmed:Stopped`, and `Warmed:Terminated`.

When an instance prepares to transition to one of the preceding steady states, Amazon EC2 Auto Scaling updates the value of the instance metadata item `autoscaling/target-lifecycle-`

`state`. To get the target lifecycle state from within the instance, you must use the Instance Metadata Service to retrieve it from the instance metadata.

> **Note**
>
> *Instance metadata* is data about an Amazon EC2 instance that applications can use to query instance information. The *Instance Metadata Service* is an on-instance component that local code uses to access instance metadata. Local code can include user data scripts or applications running on the instance.

Local code can access instance metadata from a running instance using one of two methods: Instance Metadata Service Version 1 (IMDSv1) or Instance Metadata Service Version 2 (IMDSv2). IMDSv2 uses session-oriented requests and mitigates several types of vulnerabilities that could be used to try to access the instance metadata. For details about these two methods, see Use IMDSv2 in the *Amazon EC2 User Guide*.

IMDSv2

```
[ec2-user ~]$ TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-
aws-ec2-metadata-token-ttl-seconds: 21600"` \
&& curl -H "X-aws-ec2-metadata-token: $TOKEN" -v http://169.254.169.254/latest/meta-
data/autoscaling/target-lifecycle-state
```

IMDSv1

```
[ec2-user ~]$ curl http://169.254.169.254/latest/meta-data/autoscaling/target-
lifecycle-state
```

The following is example output.

```
InService
```

The target lifecycle state is the state that the instance is transitioning to. The current lifecycle state is the state that the instance is in. These can be the same after the lifecycle action is complete and the instance finishes its transition to the target lifecycle state. You cannot retrieve the current lifecycle state of the instance from the instance metadata.

Amazon EC2 Auto Scaling started generating the target lifecycle state on March 10, 2022. If your instance transitions to one of the target lifecycle states after that date, the target lifecycle state item is present in your instance metadata. Otherwise, it is not present, and you receive an HTTP 404 error.

For more information about retrieving instance metadata, see Retrieve instance metadata in the *Amazon EC2 User Guide*.

For a tutorial that shows you how to create a lifecycle hook with a custom action in a user data script that uses the target lifecycle state, see Tutorial: Use data script and instance metadata to retrieve lifecycle state.

> **Important**
>
> To ensure that you can invoke a custom action as soon as possible, your local code should poll IMDS frequently and retry on errors.

Add lifecycle hooks to your Auto Scaling group

To put your Auto Scaling instances into a wait state and perform custom actions on them, you can add lifecycle hooks to your Auto Scaling group. Custom actions are performed as the instances launch or before they terminate. Instances remain in a wait state until you either complete the lifecycle action, or the timeout period ends.

After you create an Auto Scaling group from the AWS Management Console, you can add one or more lifecycle hooks to it, up to a total of 50 lifecycle hooks. You can also use the AWS CLI, AWS CloudFormation, or an SDK to add lifecycle hooks to an Auto Scaling group as you are creating it.

By default, when you add a lifecycle hook in the console, Amazon EC2 Auto Scaling sends lifecycle event notifications to Amazon EventBridge. Using EventBridge or a user data script is a recommended best practice. To create a lifecycle hook that sends notifications directly to Amazon SNS or Amazon SQS, you can use the put-lifecycle-hook command, as shown in the examples in this topic.

Contents

- Add lifecycle hooks (console)
- Add lifecycle hooks (AWS CLI)

Add lifecycle hooks (console)

Follow these steps to add lifecycle hooks to your Auto Scaling group. To add lifecycle hooks for scaling out (instances launching) and scaling in (instances terminating or returning to a warm pool), you must create two separate hooks.

Before you begin, confirm that you have set up a custom action, as needed, as described in Prepare to add a lifecycle hook to your Auto Scaling group.

To add a lifecycle hook for scale out

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group. A split pane opens up in the bottom of the page.
3. On the **Instance management** tab, in **Lifecycle hooks**, choose **Create lifecycle hook**.
4. To define a lifecycle hook for scale out (instances launching), do the following:
   a. For **Lifecycle hook name**, specify a name for the lifecycle hook.
   b. For **Lifecycle transition**, choose **Instance launch**.
   c. For **Heartbeat timeout**, specify the amount of time, in seconds, for instances to remain in a wait state when scaling out before the hook times out. The range is from `30` to `7200` seconds. Setting a long timeout period provides more time for your custom action to complete. Then, if you finish before the timeout period ends, send the complete-lifecycle-action command to allow the instance to proceed to the next state.
   d. For **Default result**, specify the action to take when the lifecycle hook timeout elapses or when an unexpected failure occurs. You can choose to either **CONTINUE** or **ABANDON**.
      - If you choose **CONTINUE**, the Auto Scaling group can proceed with any other lifecycle hooks and then put the instance into service.
      - If you choose **ABANDON**, the Auto Scaling group stops any remaining actions and terminates the instance immediately.
   e. (Optional) For **Notification metadata**, specify other information that you want to include when Amazon EC2 Auto Scaling sends a message to the notification target.
5. Choose **Create**.

To add a lifecycle hook for scale in

1. Choose **Create lifecycle hook** to continue where you left off after creating a lifecycle hook for scale out.
2. To define a lifecycle hook for scale in (instances terminating or returning to a warm pool), do the following:

   a. For **Lifecycle hook name**, specify a name for the lifecycle hook.

   b. For **Lifecycle transition**, choose **Instance terminate**.

   c. For **Heartbeat timeout**, specify the amount of time, in seconds, for instances to remain in a wait state when scaling out before the hook times out. We recommend a short timeout period of `30` to `120` seconds, depending on how much time you need to perform any final tasks, such as pulling EC2 logs from CloudWatch.

   d. For **Default result**, specify the action that the Auto Scaling group takes when the timeout elapses or if an unexpected failure occurs. Both **ABANDON** and **CONTINUE** let the instance terminate.

      - If you choose **CONTINUE**, the Auto Scaling group can proceed with any remaining actions, such as other lifecycle hooks, before termination.
      - If you choose **ABANDON**, the Auto Scaling group terminates the instance immediately.

   e. (Optional) For **Notification metadata**, specify other information that you want to include when Amazon EC2 Auto Scaling sends a message to the notification target.
3. Choose **Create**.

Add lifecycle hooks (AWS CLI)

Create and update lifecycle hooks using the put-lifecycle-hook command.

To perform an action on scale out, use the following command.

```
aws autoscaling put-lifecycle-hook --lifecycle-hook-name my-launch-hook  \
  --auto-scaling-group-name my-asg \
  --lifecycle-transition autoscaling:EC2_INSTANCE_LAUNCHING
```

To perform an action on scale in, use the following command instead.

```
aws autoscaling put-lifecycle-hook --lifecycle-hook-name my-termination-hook  \
```

```
  --auto-scaling-group-name my-asg \
  --lifecycle-transition autoscaling:EC2_INSTANCE_TERMINATING
```

To receive notifications using Amazon SNS or Amazon SQS, add the `--notification-target-arn` and `--role-arn` options.

The following example creates a lifecycle hook that specifies an SNS topic named *my-sns-topic* as the notification target.

```
aws autoscaling put-lifecycle-hook --lifecycle-hook-name my-termination-hook  \
  --auto-scaling-group-name my-asg \
  --lifecycle-transition autoscaling:EC2_INSTANCE_TERMINATING \
  --notification-target-arn arn:aws:sns:region:123456789012:my-sns-topic \
  --role-arn arn:aws:iam::123456789012:role/my-notification-role
```

The topic receives a test notification with the following key-value pair.

```
"Event": "autoscaling:TEST_NOTIFICATION"
```

By default, the put-lifecycle-hook command creates a lifecycle hook with a heartbeat timeout of `3600` seconds (one hour).

To change the heartbeat timeout for an existing lifecycle hook, add the `--heartbeat-timeout` option, as shown in the following example.

```
aws autoscaling put-lifecycle-hook --lifecycle-hook-name my-termination-hook \
  --auto-scaling-group-name my-asg --heartbeat-timeout 120
```

If an instance is already in a wait state, you can prevent the lifecycle hook from timing out by recording a heartbeat, using the record-lifecycle-action-heartbeat CLI command. This extends the timeout period by the timeout value specified when you created the lifecycle hook. If you finish before the timeout period ends, you can send the complete-lifecycle-action CLI command to allow the instance to proceed to the next state. For more information and examples, see Complete a lifecycle action in an Auto Scaling group.

Complete a lifecycle action in an Auto Scaling group

When an Auto Scaling group responds to a lifecycle event, it puts the instance in a wait state and sends an event notification. You can perform a custom action while the instance is in a wait state.

Completing the lifecycle action with a result of `CONTINUE` is helpful if you finish before the timeout period has expired. If you don't complete the lifecycle action, the lifecycle hook goes to the status that you specified for **Default result** after the timeout period ends.

Contents

- Complete a lifecycle action (manual)
- Complete a lifecycle action (automatic)

Complete a lifecycle action (manual)

The following procedure is for the command line interface and is not supported in the console. Information that must be replaced, such as the instance ID or the name of an Auto Scaling group, are shown in italics.

To complete a lifecycle action (AWS CLI)

1. If you need more time to complete the custom action, use the record-lifecycle-action-heartbeat command to restart the timeout period and keep the instance in a wait state. For example, if the timeout period is one hour, and you call this command after 30 minutes, the instance remains in a wait state for an additional hour, or a total of 90 minutes.

   You can specify the lifecycle action token that you received with the notification, as shown in the following command.

   ```
   aws autoscaling record-lifecycle-action-heartbeat --lifecycle-hook-name my-launch-hook \
     --auto-scaling-group-name my-asg --lifecycle-action-token bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635
   ```

   Alternatively, you can specify the ID of the instance that you received with the notification, as shown in the following command.

   ```
   aws autoscaling record-lifecycle-action-heartbeat --lifecycle-hook-name my-launch-hook \
     --auto-scaling-group-name my-asg --instance-id i-1a2b3c4d
   ```

2. If you finish the custom action before the timeout period ends, use the complete-lifecycle-action command so that the Auto Scaling group can continue launching or terminating the instance. You can specify the lifecycle action token, as shown in the following command.

```
aws autoscaling complete-lifecycle-action --lifecycle-action-result CONTINUE \
  --lifecycle-hook-name my-launch-hook --auto-scaling-group-name my-asg \
  --lifecycle-action-token bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635
```

Alternatively, you can specify the ID of the instance, as shown in the following command.

```
aws autoscaling complete-lifecycle-action --lifecycle-action-result CONTINUE \
  --instance-id i-1a2b3c4d --lifecycle-hook-name my-launch-hook \
  --auto-scaling-group-name my-asg
```

Complete a lifecycle action (automatic)

If you have a user data script that configures your instances after they launch, you do not need to manually complete lifecycle actions. You can add the complete-lifecycle-action command to the script. The script can retrieve the instance ID from the instance metadata and signal Amazon EC2 Auto Scaling when the bootstrap scripts have completed successfully.

If you are not doing so already, update your script to retrieve the instance ID of the instance from the instance metadata. For more information, see Retrieve instance metadata in the *Amazon EC2 User Guide*.

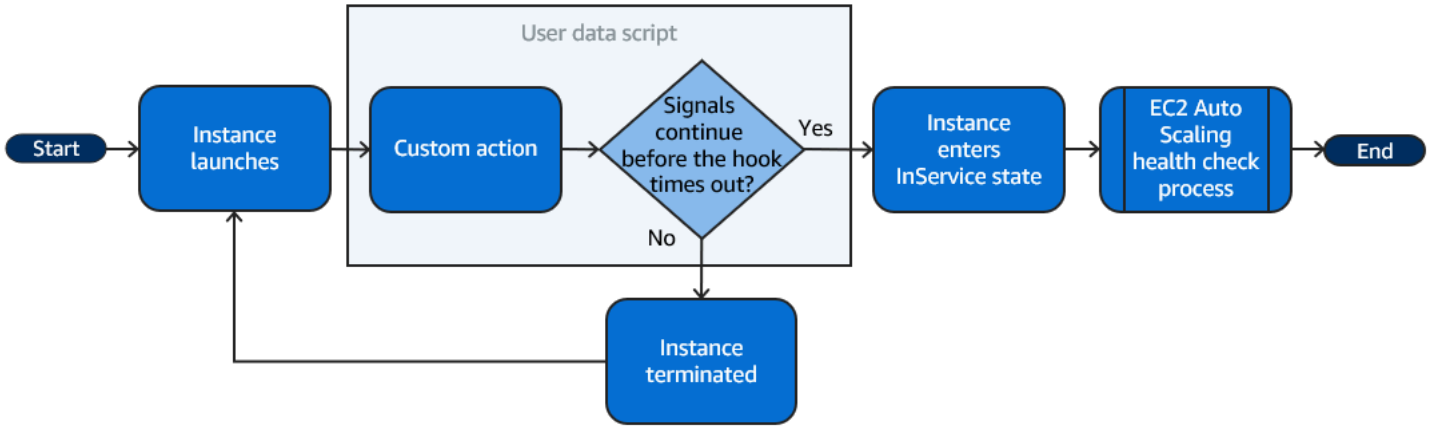
If you use Lambda, you can also set up a callback in your function's code to let the lifecycle of the instance proceed if the custom action is successful. For more information, see Tutorial: Configure a lifecycle hook that invokes a Lambda function.

Tutorial: Use data script and instance metadata to retrieve lifecycle state

A common way to create custom actions for lifecycle hooks is to use notifications that Amazon EC2 Auto Scaling sends to other services, such as Amazon EventBridge. However, you can avoid having to create additional infrastructure by instead using a user data script to move the code that configures instances and completes the lifecycle action into the instances themselves.

The following tutorial shows you how to get started using a user data script and instance metadata. You create a basic Auto Scaling group configuration with a user data script that reads the target lifecycle state of the instances in your group and performs a callback action at a specific phase of an instance's lifecycle to continue the launch process.

The following illustration summarizes the flow for a scale-out event when you use a user data script to perform a custom action. After an instance launches, the lifecycle of the instance is paused until the lifecycle hook is completed, either by timing out or by Amazon EC2 Auto Scaling receiving a signal to continue.



Contents

- Step 1: Create an IAM role with permissions to complete lifecycle actions
- Step 2: Create a launch template and include the IAM role and a user data script
- Step 3: Create an Auto Scaling group
- Step 4: Add a lifecycle hook
- Step 5: Test and verify the functionality
- Step 6: Clean up
- Related resources

Step 1: Create an IAM role with permissions to complete lifecycle actions

When you use the AWS CLI or an AWS SDK to send a callback to complete lifecycle actions, you must use an IAM role with permissions to complete lifecycle actions.

To create the policy

1. Open the Policies page of the IAM console, and then choose **Create policy**.
2. Choose the **JSON** tab.
3. In the **Policy Document** box, copy and paste the following policy document into the box. Replace the *`sample text`* with your account number and the name of the Auto Scaling group that you want to create (**`TestAutoScalingEvent-group`**).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CompleteLifecycleAction"
      ],
      "Resource":
 "arn:aws:autoscaling:*:123456789012:autoScalingGroup:*:autoScalingGroupName/
TestAutoScalingEvent-group"
    }
  ]
}
```

4. Choose **Next**.
5. For **Policy name**, enter **`TestAutoScalingEvent-policy`**. Choose **Create policy**.

When you finish creating the policy, you can create a role that uses it.

To create the role

1. In the navigation pane on the left, choose **Roles**.
2. Choose **Create role**.
3. For **Select trusted entity**, choose **AWS service**.
4. For your use case, choose **EC2** and then choose **Next**.
5. Under **Add permissions**, choose the policy that you created (**TestAutoScalingEvent-policy**). Then, choose **Next**.
6. On the **Name, review, and create** page, for **Role name**, enter **`TestAutoScalingEvent-role`** and choose **Create role**.

Step 2: Create a launch template and include the IAM role and a user data script

Create a launch template to use with your Auto Scaling group. Include the IAM role you created and the provided sample user data script.

To create a launch template

1. Open the Launch templates page of the Amazon EC2 console.
2. Choose **Create launch template**.
3. For **Launch template name**, enter **`TestAutoScalingEvent-template`**.
4. Under **Auto Scaling guidance**, select the check box.
5. For **Application and OS Images (Amazon Machine Image)**, choose Amazon Linux 2 (HVM), SSD Volume Type, 64-bit (x86) from the **Quick Start** list.
6. For **Instance type**, choose a type of Amazon EC2 instance (for example, "t2.micro").
7. For **Advanced details**, expand the section to view the fields.
8. For **IAM instance profile**, choose the IAM instance profile name of your IAM role (**TestAutoScalingEvent-role**). An instance profile is a container for an IAM role that allows Amazon EC2 to pass the IAM role to an instance when the instance is launched.

   When you used the IAM console to create an IAM role, the console automatically created an instance profile with the same name as its corresponding role.
9. For **User data**, copy and paste the following sample user data script into the field. Replace the sample text for `group_name` with the name of the Auto Scaling group that you want to create and `region` with the AWS Region you want your Auto Scaling group to use.

```
#!/bin/bash

function get_target_state {
    echo $(curl -s http://169.254.169.254/latest/meta-data/autoscaling/target-
lifecycle-state)
}

function get_instance_id {
    echo $(curl -s http://169.254.169.254/latest/meta-data/instance-id)
}

function complete_lifecycle_action {
    instance_id=$(get_instance_id)
    group_name='TestAutoScalingEvent-group'
    region='us-west-2'

    echo $instance_id
    echo $region
    echo $(aws autoscaling complete-lifecycle-action \
```

```
      --lifecycle-hook-name TestAutoScalingEvent-hook \
      --auto-scaling-group-name $group_name \
      --lifecycle-action-result CONTINUE \
      --instance-id $instance_id \
      --region $region)
}

function main {
    while true
    do
        target_state=$(get_target_state)
        if [ \"$target_state\" = \"InService\" ]; then
            # Change hostname
            export new_hostname="${group_name}-$instance_id"
            hostname $new_hostname
            # Send callback
            complete_lifecycle_action
            break
        fi
        echo $target_state
        sleep 5
    done
}

main
```

This simple user data script does the following:

- Calls the instance metadata to retrieve the target lifecycle state and instance ID from the instance metadata
- Retrieves the target lifecycle state repeatedly until it changes to `InService`
- Changes the hostname of the instance to the instance ID prepended with the name of the Auto Scaling group, if the target lifecycle state is `InService`
- Sends a callback by calling the **complete-lifecycle-action** CLI command to signal Amazon EC2 Auto Scaling to `CONTINUE` the EC2 launch process

10. Choose **Create launch template**.
11. On the confirmation page, choose **Create Auto Scaling group**.

> **Note**
>
> For other examples that you can use as a reference for developing your user data script, see the GitHub repository for Amazon EC2 Auto Scaling.

Step 3: Create an Auto Scaling group

After you create your launch template, create an Auto Scaling group.

To create an Auto Scaling group

1. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter a name for your Auto Scaling group (**`TestAutoScalingEvent-group`**).
2. Choose **Next** to go to the **Choose instance launch options** page.
3. For **Network**, choose a VPC.
4. For **Availability Zones and subnets**, choose one or more subnets from one or more Availability Zones.
5. In the **Instance type requirements** section, use the default setting to simplify this step. (Do not override the launch template.) For this tutorial, you will launch only one On-Demand Instance using the instance type specified in your launch template.
6. Choose **Skip to review** at the bottom of the screen.
7. On the **Review** page, review the details of your Auto Scaling group, and then choose **Create Auto Scaling group**.

Step 4: Add a lifecycle hook

Add a lifecycle hook to hold the instance in a wait state until your lifecycle action is complete.

To add a lifecycle hook

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group. A split pane opens up in the bottom of the page.
3. In the lower pane, on the **Instance management** tab, in **Lifecycle hooks**, choose **Create lifecycle hook**.

4. To define a lifecycle hook for scale out (instances launching), do the following:
   a. For **Lifecycle hook name**, enter **`TestAutoScalingEvent-hook`**.
   b. For **Lifecycle transition**, choose **Instance launch**.
   c. For **Heartbeat timeout**, enter **`300`** for the number of seconds to wait for a callback from your user data script.
   d. For **Default result**, choose **ABANDON**. If the hook times out without receiving a callback from your user data script, the Auto Scaling group terminates the new instance.
   e. (Optional) Keep **Notification metadata** blank.
5. Choose **Create**.

Step 5: Test and verify the functionality

To test the functionality, update the Auto Scaling group by increasing the desired capacity of the Auto Scaling group by 1. The user data script runs and starts to check the instance's target lifecycle state soon after the instance launches. The script changes the hostname and sends a callback action when the target lifecycle state is `InService`. This usually takes only a few seconds to finish.

To increase the size of the Auto Scaling group

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group. View details in a lower pane while still seeing the top rows of the upper pane.
3. In the lower pane, on the **Details** tab, choose **Group details**, **Edit**.
4. For **Desired capacity**, increase the current value by 1.
5. Choose **Update**. While the instance is being launched, the **Status** column in the upper pane displays a status of *Updating capacity*.

After increasing the desired capacity, you can verify that your instance has successfully launched and is not terminated from the description of scaling activities.

To view the scaling activity

1. Return to the **Auto Scaling groups** page and select your group.
2. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched an instance.

3. If the user data script fails, after the timeout period passes, you see a scaling activity with a status of `Canceled` and a status message of `Instance failed to complete user's Lifecycle Action: Lifecycle Action with token e85eb647-4fe0-4909-b341-a6c42EXAMPLE was abandoned: Lifecycle Action Completed with ABANDON Result`.

Step 6: Clean up

If you are done working with the resources that you created for this tutorial, use the following steps to delete them.

To delete the lifecycle hook

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group.
3. On the **Instance management** tab, in **Lifecycle hooks**, choose the lifecycle hook (`TestAutoScalingEvent-hook`).
4. Choose **Actions**, **Delete**.
5. Choose **Delete** again to confirm.

To delete the launch template

1. Open the Launch templates page of the Amazon EC2 console.
2. Select your launch template (`TestAutoScalingEvent-template`) and then choose **Actions**, **Delete template**.
3. When prompted for confirmation, type **`Delete`** to confirm deleting the specified launch template and then choose **Delete**.

If you are done working with the example Auto Scaling group, delete it. You can also delete the IAM role and permissions policy that you created.

To delete the Auto Scaling group

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group (`TestAutoScalingEvent-group`) and choose **Delete**.

3. When prompted for confirmation, type **`delete`** to confirm deleting the specified Auto Scaling group and then choose **Delete**.

   A loading icon in the **Name** column indicates that the Auto Scaling group is being deleted. It takes a few minutes to terminate the instances and delete the group.

To delete the IAM role

1. Open the Roles page of the IAM console.
2. Select the function's role (`TestAutoScalingEvent-role`).
3. Choose **Delete**.
4. When prompted for confirmation, type the name of the role and then choose **Delete**.

To delete the IAM policy

1. Open the Policies page of the IAM console.
2. Select the policy that you created (`TestAutoScalingEvent-policy`).
3. Choose **Actions**, **Delete**.
4. When prompted for confirmation, type the name of the policy and then choose **Delete**.

Related resources

The following related topics can be helpful as you develop code that invokes actions on instances based on data available in the instance metadata.

- Retrieve the target lifecycle state through instance metadata. This section describes the lifecycle state for other use cases, such as instance termination.
- Add lifecycle hooks (console). This procedure shows you how to add lifecycle hooks for both scale out (instances launching) and scale in (instances terminating or returning to a warm pool).
- Instance metadata categories in the *Amazon EC2 User Guide*. This topic lists all categories of instance metadata that you can use to invoke actions on EC2 instances.

For a tutorial that shows you how to use Amazon EventBridge to create rules that invoke Lambda functions based on events that happen to the instances in your Auto Scaling group, see Tutorial: Configure a lifecycle hook that invokes a Lambda function.

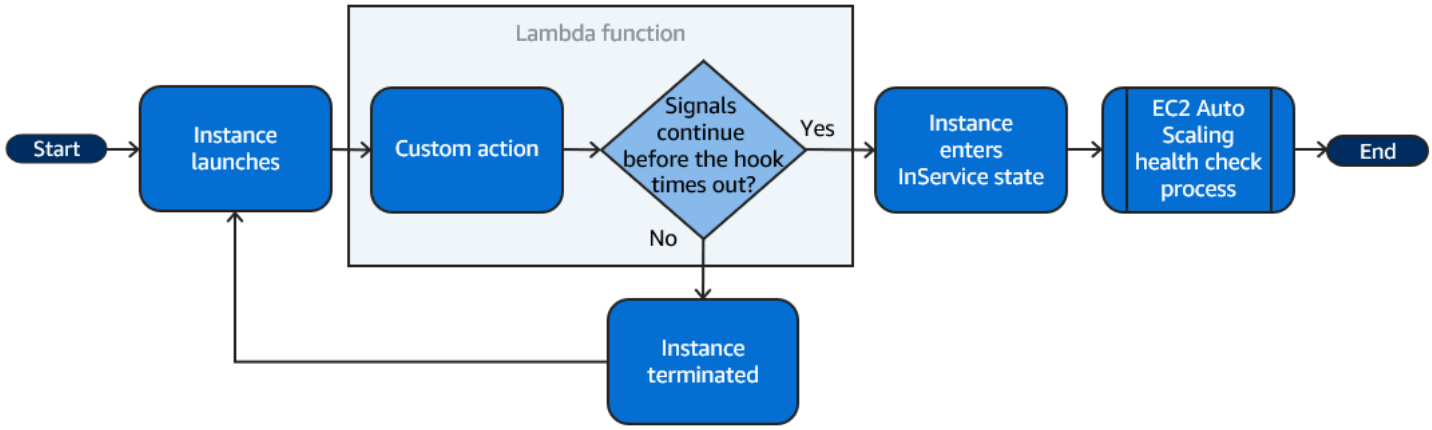
Tutorial: Configure a lifecycle hook that invokes a Lambda function

In this exercise, you create an Amazon EventBridge rule that includes a filter pattern that when matched, invokes an AWS Lambda function as the rule target. We provide the filter pattern and sample function code to use.

If everything is configured correctly, at the end of this tutorial, the Lambda function performs a custom action when instances launch. The custom action simply logs the event in the CloudWatch Logs log stream associated with the Lambda function.

The Lambda function also performs a callback to let the lifecycle of the instance proceed if this action is successful, but lets the instance abandon the launch and terminate if the action fails.

The following illustration summarizes the flow for a scale-out event when you use a Lambda function to perform a custom action. After an instance launches, the lifecycle of the instance is paused until the lifecycle hook is completed, either by timing out or by Amazon EC2 Auto Scaling receiving a signal to continue.



Contents

- Prerequisites
- Step 1: Create an IAM role with permissions to complete lifecycle actions
- Step 2: Create a Lambda function
- Step 3: Create an EventBridge rule
- Step 4: Add a lifecycle hook
- Step 5: Test and verify the event
- Step 6: Clean up
- Related resources

Prerequisites

Before you begin this tutorial, create an Auto Scaling group, if you don't have one already. To create an Auto Scaling group, open the Auto Scaling groups page of the Amazon EC2 console and choose **Create Auto Scaling group**.

Step 1: Create an IAM role with permissions to complete lifecycle actions

Before you create a Lambda function, you must first create an execution role and a permissions policy to allow Lambda to complete lifecycle hooks.

To create the policy

1. Open the Policies page of the IAM console, and then choose **Create policy**.
2. Choose the **JSON** tab.
3. In the **Policy Document** box, paste the following policy document into the box, replacing the text in *italics* with your account number and the name of your Auto Scaling group.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CompleteLifecycleAction"
      ],
      "Resource":
 "arn:aws:autoscaling:*:123456789012:autoScalingGroup:*:autoScalingGroupName/my-
asg"
    }
  ]
}
```

4. Choose **Next**.
5. For **Policy name**, enter **`LogAutoScalingEvent-policy`**. Choose **Create policy**.

When you finish creating the policy, you can create a role that uses it.

To create the role

1. In the navigation pane on the left, choose **Roles**.

2. Choose **Create role**.
3. For **Select trusted entity**, choose **AWS service**.
4. For your use case, choose **Lambda** and then choose **Next**.
5. Under **Add permissions**, choose the policy that you created (**LogAutoScalingEvent-policy**) and the policy named **AWSLambdaBasicExecutionRole**. Then, choose **Next**.

> **Note**
>
> The **AWSLambdaBasicExecutionRole** policy has the permissions that the function needs to write logs to CloudWatch Logs.

6. On the **Name, review, and create** page, for **Role name**, enter **`LogAutoScalingEvent-role`** and choose **Create role**.

Step 2: Create a Lambda function

Create a Lambda function to serve as the target for events. The sample Lambda function, written in Node.js, is invoked by EventBridge when a matching event is emitted by Amazon EC2 Auto Scaling.

To create a Lambda function

1. Open the Functions page on the Lambda console.
2. Choose **Create function**, **Author from scratch**.
3. Under **Basic information**, for **Function name**, enter **`LogAutoScalingEvent`**.
4. For **Runtime**, choose **Node.js 18.x**.
5. Scroll down and choose **Change default execution role**, and then for **Execution role**, choose **Use an existing role**.
6. For **Existing role**, choose **LogAutoScalingEvent-role**.
7. Leave the other default values.
8. Choose **Create function**. You are returned to the function's code and configuration.
9. With your `LogAutoScalingEvent` function still open in the console, under **Code source**, in the editor, paste the following sample code into the file named index.mjs.

```
import { AutoScalingClient, CompleteLifecycleActionCommand } from "@aws-sdk/client-
auto-scaling";
```

```
export const handler = async(event) => {
  console.log('LogAutoScalingEvent');
  console.log('Received event:', JSON.stringify(event, null, 2));
  var autoscaling = new AutoScalingClient({ region: event.region });
  var eventDetail = event.detail;
  var params = {
    AutoScalingGroupName: eventDetail['AutoScalingGroupName'], /* required */
    LifecycleActionResult: 'CONTINUE', /* required */
    LifecycleHookName: eventDetail['LifecycleHookName'], /* required */
    InstanceId: eventDetail['EC2InstanceId'],
    LifecycleActionToken: eventDetail['LifecycleActionToken']
  };
  var response;
  const command = new CompleteLifecycleActionCommand(params);
  try {
    var data = await autoscaling.send(command);
    console.log(data); // successful response
    response = {
      statusCode: 200,
      body: JSON.stringify('SUCCESS'),
    };
  } catch (err) {
    console.log(err, err.stack); // an error occurred
    response = {
      statusCode: 500,
      body: JSON.stringify('ERROR'),
    };
  }
  return response;
};
```

This code simply logs the event so that, at the end of this tutorial, you can see an event appear in the CloudWatch Logs log stream that's associated with this Lambda function.

10. Choose **Deploy**.

Step 3: Create an EventBridge rule

Create an EventBridge rule to run your Lambda function. For more information about using EventBridge, see Use EventBridge to handle Auto Scaling events.

To create a rule using the console

1. Open the EventBridge console.
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. For **Define rule detail**, do the following:
   a. For **Name**, enter **`LogAutoScalingEvent-rule`**.
   b. For **Event bus**, choose **default**. When an AWS service in your account generates an event, it always goes to your account's default event bus.
   c. For **Rule type**, choose **Rule with an event pattern**.
   d. Choose **Next**.
5. For **Build event pattern**, do the following:
   a. For **Event source**, choose **AWS events or EventBridge partner events**.
   b. Scroll down to **Event pattern**, and do the following:
   c. i. For **Event source**, choose **AWS services**.
      ii. For **AWS service**, choose **Auto Scaling**.
      iii. For **Event type**, choose **Instance Launch and Terminate**.
      iv. By default, the rule matches any scale-in or scale-out event. To create a rule that notifies you when there is a scale-out event and an instance is put into a wait state due to a lifecycle hook, choose **Specific instance event(s)** and select **EC2 Instance-launch Lifecycle Action**.
      v. By default, the rule matches any Auto Scaling group in the Region. To make the rule match a specific Auto Scaling group, choose **Specific group name(s)** and select the group.
      vi. Choose **Next**.
6. For **Select target(s)**, do the following:
   a. For **Target types**, choose **AWS service**.
   b. For **Select a target**, choose **Lambda function**.
   c. For **Function**, choose **LogAutoScalingEvent**.
   d. Choose **Next** twice.
7. On the **Review and create** page, choose **Create rule**.

Step 4: Add a lifecycle hook

In this section, you add a lifecycle hook so that Lambda runs your function on instances at launch.

To add a lifecycle hook

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group. A split pane opens up in the bottom of the page.
3. In the lower pane, on the **Instance management** tab, in **Lifecycle hooks**, choose **Create lifecycle hook**.
4. To define a lifecycle hook for scale out (instances launching), do the following:
   a. For **Lifecycle hook name**, enter **`LogAutoScalingEvent-hook`**.
   b. For **Lifecycle transition**, choose **Instance launch**.
   c. For **Heartbeat timeout**, enter **`300`** for the number of seconds to wait for a callback from your Lambda function.
   d. For **Default result**, choose **ABANDON**. This means that the Auto Scaling group will terminate a new instance if the hook times out without receiving a callback from your Lambda function.
   e. (Optional) Leave **Notification metadata** empty. The event data that we pass to EventBridge contains all of the necessary information to invoke the Lambda function.
5. Choose **Create**.

Step 5: Test and verify the event

To test the event, update the Auto Scaling group by increasing the desired capacity of the Auto Scaling group by 1. Your Lambda function is invoked within a few seconds after increasing the desired capacity.

To increase the size of the Auto Scaling group

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group to view details in a lower pane and still see the top rows of the upper pane.
3. In the lower pane, on the **Details** tab, choose **Group details**, **Edit**.

4. For **Desired capacity**, increase the current value by 1.
5. Choose **Update**. While the instance is being launched, the **Status** column in the upper pane displays a status of *Updating capacity*.

After increasing the desired capacity, you can verify that your Lambda function was invoked.

To view the output from your Lambda function

1. Open the Log groups page of the CloudWatch console.
2. Select the name of the log group for your Lambda function (`/aws/lambda/LogAutoScalingEvent`).
3. Select the name of the log stream to view the data provided by the function for the lifecycle action.

Next, you can verify that your instance has successfully launched from the description of scaling activities.

To view the scaling activity

1. Return to the **Auto Scaling groups** page and select your group.
2. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched an instance.
   - If the action was successful, the scaling activity will have a status of "Successful".
   - If it failed, after waiting a few minutes, you will see a scaling activity with a status of "Cancelled" and a status message of "Instance failed to complete user's Lifecycle Action: Lifecycle Action with token e85eb647-4fe0-4909-b341-a6c42EXAMPLE was abandoned: Lifecycle Action Completed with ABANDON Result".

To decrease the size of the Auto Scaling group

If you do not need the additional instance that you launched for this test, you can open the **Details** tab and decrease **Desired capacity** by 1.

Step 6: Clean up

If you are done working with the resources that you created just for this tutorial, use the following steps to delete them.

To delete the lifecycle hook

1. Open the Auto Scaling groups page of the Amazon EC2 console.
2. Select the check box next to your Auto Scaling group.
3. On the **Instance management** tab, in **Lifecycle hooks**, choose the lifecycle hook (`LogAutoScalingEvent-hook`).
4. Choose **Actions**, **Delete**.
5. Choose **Delete** again to confirm.

To delete the Amazon EventBridge rule

1. Open the Rules page in the Amazon EventBridge console.
2. Under **Event bus**, choose the event bus that is associated with the rule (`Default`).
3. Select the check box next to your rule (`LogAutoScalingEvent-rule`).
4. Choose **Delete**.
5. When prompted for confirmation, type the name of the rule and then choose **Delete**.

If you are done working with the example function, delete it. You can also delete the log group that stores the function's logs, and the execution role and permissions policy that you created.

To delete a Lambda function

1. Open the Functions page on the Lambda console.
2. Choose the function (`LogAutoScalingEvent`).
3. Choose **Actions**, **Delete**.
4. When prompted for confirmation, type **`delete`** to confirm deleting the specified function and then choose **Delete**.

To delete the log group

1. Open the Log groups page of the CloudWatch console.

2. Select the function's log group (`/aws/lambda/LogAutoScalingEvent`).
3. Choose **Actions**, **Delete log group(s)**.
4. In the **Delete log group(s)** dialog box, choose **Delete**.

To delete the execution role

1. Open the Roles page of the IAM console.
2. Select the function's role (`LogAutoScalingEvent-role`).
3. Choose **Delete**.
4. When prompted for confirmation, type the name of the role and then choose **Delete**.

To delete the IAM policy

1. Open the Policies page of the IAM console.
2. Select the policy that you created (`LogAutoScalingEvent-policy`).
3. Choose **Actions**, **Delete**.
4. When prompted for confirmation, type the name of the policy and then choose **Delete**.

Related resources

The following related topics can be helpful as you create EventBridge rules based on events that happen to the instances in your Auto Scaling group.

- Use EventBridge to handle Auto Scaling events. This section shows you examples of events for other use cases, including events for scale in.
- Add lifecycle hooks (console). This procedure shows you how to add lifecycle hooks for both scale out (instances launching) and scale in (instances terminating or returning to a warm pool).

For a tutorial that shows you how to use the Instance Metadata Service (IMDS) to invoke an action from within the instance itself, see Tutorial: Use data script and instance metadata to retrieve lifecycle state.

Decrease latency for applications with long boot times using warm pools

A warm pool gives you the ability to decrease latency for your applications that have exceptionally long boot times, for example, because instances need to write massive amounts of data to disk. With warm pools, you no longer have to over-provision your Auto Scaling groups to manage latency in order to improve application performance. For more information, see the following blog post Scaling your applications faster with EC2 Auto Scaling Warm Pools.

> **Important**
>
> Creating a warm pool when it's not required can lead to unnecessary costs. If your first boot time does not cause noticeable latency issues for your application, there probably isn't a need for you to use a warm pool.

Topics

- Core concepts
- Prerequisites
- Update the instances in a warm pool
- Related resources
- Limitations
- Use lifecycle hooks with a warm pool in Auto Scaling group
- Create a warm pool for an Auto Scaling group
- View health check status and the reason for health check failures
- Examples for creating and managing warm pools with the AWS CLI

Core concepts

Before you get started, familiarize yourself with the following core concepts:

Warm pool

A warm pool is a pool of pre-initialized EC2 instances that sits alongside an Auto Scaling group. Whenever your application needs to scale out, the Auto Scaling group can draw on the warm

pool to meet its new desired capacity. This helps you to ensure that instances are ready to quickly start serving application traffic, accelerating the response to a scale-out event. As instances leave the warm pool, they count toward the desired capacity of the group. This is known as a *warm start*.

While instances are in the warm pool, your scaling policies only scale out if the metric value from instances that are in the `InService` state is greater than the scaling policy's alarm high threshold (which is the same as the target utilization of a target tracking scaling policy).

Warm pool size

By default, the size of the warm pool is calculated as the difference between the Auto Scaling group's maximum capacity and its desired capacity. For example, if the desired capacity of your Auto Scaling group is 6 and the maximum capacity is 10, the size of your warm pool will be 4 when you first set up the warm pool and the pool is initializing.

To specify the warm pool's maximum capacity separately, use the custom specification (`MaxGroupPreparedCapacity`) option and set a custom value for it that is greater than the current capacity of the group. If you provide a custom value, the size of the warm pool is calculated as the difference between the custom value and the current desired capacity of the group. For example, if the desired capacity of your Auto Scaling group is 6, if the maximum capacity is 20, and if the custom value is 8, the size of your warm pool will be 2 when you first set up the warm pool and the pool is initializing.

You might only need to use the custom specification (`MaxGroupPreparedCapacity`) option when working with large Auto Scaling groups to manage the cost benefits of having a warm pool. For example, an Auto Scaling group with 1,000 instances, a maximum capacity of 1,500 (to provide extra capacity for emergency traffic spikes), and a warm pool of 100 instances might help you achieve your goals better than keeping 500 instances reserved for future use inside the warm pool.

Minimum warm pool size

Consider using the minimum size setting to statically set the minimum number of instances to maintain in the warm pool. There is no minimum size set by default.

Warm pool instance state

You can keep instances in the warm pool in one of three states: `Stopped`, `Running`, or `Hibernated`. Keeping instances in a `Stopped` state is an effective way to minimize costs. With stopped instances, you pay only for the volumes that you use and the Elastic IP addresses attached to the instances.

Alternatively, you can keep instances in a `Hibernated` state to stop instances without deleting their memory contents (RAM). When an instance is hibernated, this signals the operating system to save the contents of your RAM to your Amazon EBS root volume. When the instance is started again, the root volume is restored to its previous state and the RAM contents are reloaded. While the instances are in hibernation, you pay only for the EBS volumes, including storage for the RAM contents, and the Elastic IP addresses attached to the instances.

Keeping instances in a `Running` state inside the warm pool is also possible, but is highly discouraged to avoid incurring unnecessary charges. When instances are stopped or hibernated, you are saving the cost of the instances themselves. You pay for the instances only when they are running.

Lifecycle hooks

You use lifecycle hooks to put instances into a wait state so that you can perform custom actions on the instances. Custom actions are performed as the instances launch or before they terminate.

In a warm pool configuration, lifecycle hooks delay instances from being stopped or hibernated and from being put in service during a scale-out event until they have finished initializing. If you add a warm pool to your Auto Scaling group without a lifecycle hook, instances that take a long time to finish initializing could be stopped or hibernated and then put in service during a scale-out event before they are ready.

Instance reuse policy

By default, Amazon EC2 Auto Scaling terminates your instances when your Auto Scaling group scales in. Then, it launches new instances into the warm pool to replace the instances that were terminated.

If you want to return instances to the warm pool instead, you can specify an instance reuse policy. This lets you reuse instances that are already configured to serve application traffic. To make sure that your warm pool is not over-provisioned, Amazon EC2 Auto Scaling can terminate instances in the warm pool to reduce its size when it is larger than necessary based on its settings. When terminating instances in the warm pool, it uses the default termination policy to choose which instances to terminate first.

> **Important**
>
> If you want to hibernate instances on scale in and there are existing instances in the Auto Scaling group, they must meet the requirements for instance hibernation. If they

don't, when instances return to the warm pool, they will fallback to being stopped instead of being hibernated.

> **Note**
>
> Currently, you can only specify an instance reuse policy by using the AWS CLI or an SDK. This feature is not available from the console.

Prerequisites

Before you create a warm pool for your Auto Scaling group, decide how you will use lifecycle hooks to initialize new instances with an appropriate initial state.

To perform custom actions on instances while they are in a wait state due to a lifecycle hook, you have two options:

- For simple scenarios where you want to run commands on your instances at launch, you can include a user data script when you create a launch template or launch configuration for your Auto Scaling group. User data scripts are just normal shell scripts or cloud-init directives that are run by cloud-init when your instances start. The script can also control when your instances transition to the next state by using the ID of the instance on which it runs. If you are not doing so already, update your script to retrieve the instance ID of the instance from the instance metadata. For more information, see Retrieve instance metadata in the *Amazon EC2 User Guide*.

  > **Tip**
  >
  > To run user data scripts when an instance restarts, the user data must be in the MIME multi-part format and specify the following in the `#cloud-config` section of the user data:
  >
  > ```
  > #cloud-config
  > cloud_final_modules:
  > - [scripts-user, always]
  > ```

- For advanced scenarios where you need a service such as AWS Lambda to do something as instances are entering or leaving the warm pool, you can create a lifecycle hook for your Auto

Scaling group and configure the target service to perform custom actions based on lifecycle notifications. For more information, see Supported notification targets.

Prepare instances for hibernation

To prepare Auto Scaling instances to use the `Hibernated` pool state, create a new launch template or launch configuration that is set up correctly to support instance hibernation, as described in the Hibernation prerequisites topic in the *Amazon EC2 User Guide*. Then, associate the new launch template or launch configuration with the Auto Scaling group and start an instance refresh to replace the instances associated with a previous launch template or launch configuration. For more information, see Use an instance refresh to update instances in an Auto Scaling group.

Update the instances in a warm pool

To update the instances in a warm pool, you create a new launch template or launch configuration and associate it with the Auto Scaling group. Any new instances are launched using the new AMI and other updates that are specified in the launch template or launch configuration, but existing instances are not affected.

To force replacement warm pool instances to launch that use the new launch template or launch configuration, you can start an instance refresh to do a rolling update of your group. An instance refresh first replaces `InService` instances. Then it replaces instances in the warm pool. For more information, see Use an instance refresh to update instances in an Auto Scaling group.

Related resources

You can visit our GitHub repository for examples of lifecycle hooks for warm pools.

Limitations

- You can't add a warm pool to an Auto Scaling group that has a mixed instances policy. You also can't add a warm pool to an Auto Scaling group that has launch template or launch configuration that requests Spot Instances.
- Amazon EC2 Auto Scaling can put an instance in a `Stopped` or `Hibernated` state only if it has an Amazon EBS volume as its root device. Instances that use instance stores for the root device cannot be stopped or hibernated.

- Amazon EC2 Auto Scaling can put an instance in a `Hibernated` state only if meets all of the requirements listed in the Hibernation prerequisites topic in the *Amazon EC2 User Guide*.
- If your warm pool is depleted when there is a scale-out event, instances will launch directly into the Auto Scaling group (a *cold start*). You could also experience cold starts if an Availability Zone is out of capacity.
- If an instance within the warm pool encounters an issue during the launch process, preventing it from reaching the `InService` state, the instance will be considered a failed launch and terminated. This applies regardless of the underlying cause, such as an insufficient capacity error or any other factor.
- If you try using a warm pool with an Amazon Elastic Kubernetes Service (Amazon EKS) managed node group, instances that are still initializing might register with your Amazon EKS cluster. As a result, the cluster might schedule jobs on an instance as it is preparing to be stopped or hibernated.
- Likewise, if you try using a warm pool with an Amazon ECS cluster, instances might register with the cluster before they finish initializing. To solve this problem, you must configure a launch template or launch configuration that includes a special agent configuration variable in the user data. For more information, see Using a warm pool for your Auto Scaling group in the *Amazon Elastic Container Service Developer Guide*.
- Hibernation support for warm pools is available in all commercial AWS Regions where Amazon EC2 Auto Scaling and hibernation are available, except the following:
  - Asia Pacific (Hyderabad)
  - Asia Pacific (Melbourne)
  - Canada West (Calgary)
  - China (Beijing) Region
  - China (Ningxia) Region
  - Europe (Spain)
  - Israel (Tel Aviv)

Use lifecycle hooks with a warm pool in Auto Scaling group

Instances in a warm pool maintain their own independent lifecycle to help you create the appropriate custom action for each transition. This lifecycle is designed to help you to invoke actions in a target service (for example, a Lambda function) while an instance is still initializing and before it is put in service.

> **Note**
>
> The API operations that you use to add and manage lifecycle hooks and complete lifecycle actions are not changed. Only the instance lifecycle is changed.

For more information about adding a lifecycle hook, see Add lifecycle hooks to your Auto Scaling group. For more information about completing a lifecycle action, see Complete a lifecycle action in an Auto Scaling group.

For instances entering the warm pool, you might need a lifecycle hook for one of the following reasons:

- You want to launch EC2 instances from an AMI that takes a long time to finish initializing.
- You want to run user data scripts to bootstrap the EC2 instances.

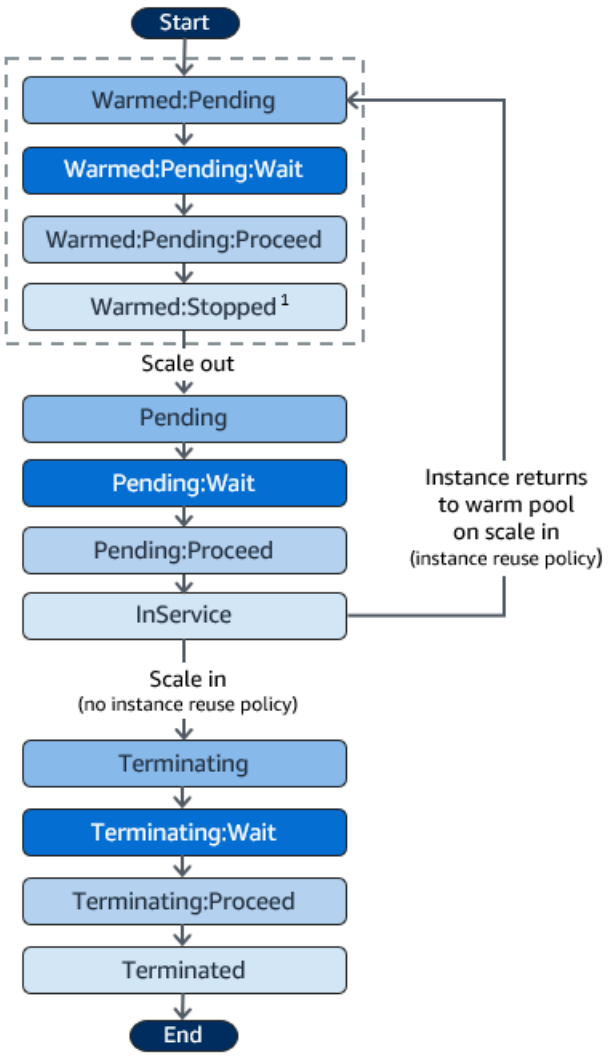
For instances leaving the warm pool, you might need a lifecycle hook for one of the following reasons:

- You can use some extra time to prepare EC2 instances for use. For example, you might have services that must start when an instance restarts before your application can work correctly.
- You want to pre-populate cache data so that a new server doesn't launch with an empty cache.
- You want to register new instances as managed instances with your configuration management service.

Lifecycle state transitions for instances in a warm pool

An Auto Scaling instance can transition through many states as part of its lifecycle.

The following diagram shows the transition between Auto Scaling states when you use a warm pool:



[1] This state varies based on the warm pool's pool state setting. If the pool state is set to `Running`, then this state is `Warmed:Running` instead. If the pool state is set to `Hibernated`, then this state is `Warmed:Hibernated` instead.

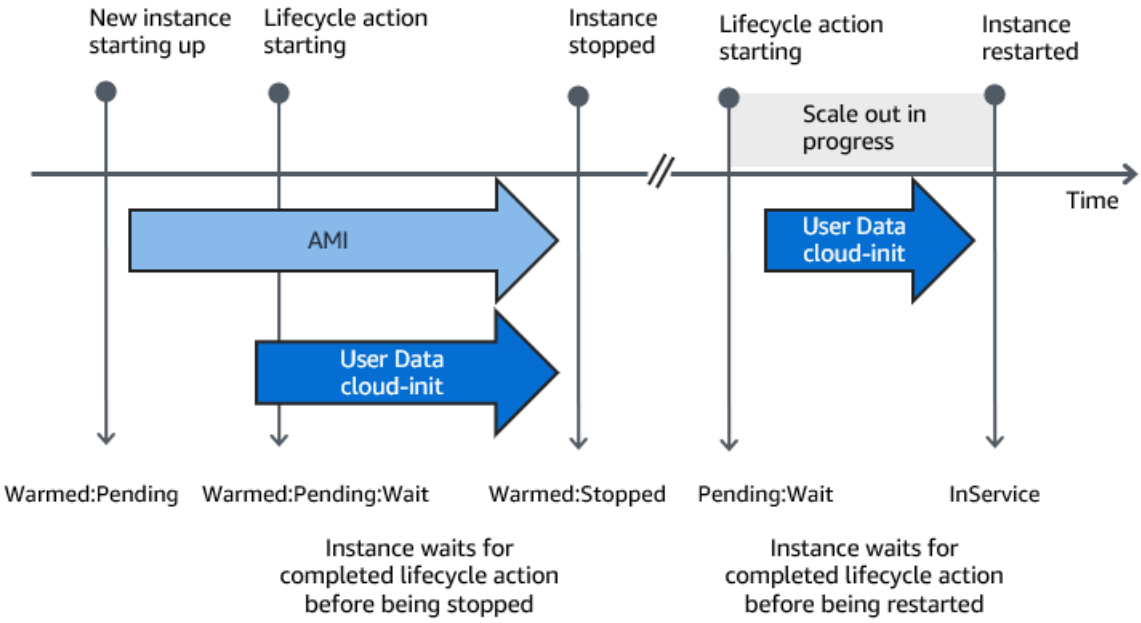
When you add lifecycle hooks, consider the following:

- When a lifecycle hook is configured for the `autoscaling:EC2_INSTANCE_LAUNCHING` lifecycle action, a newly launched instance first pauses to perform a custom action when it reaches the `Warmed:Pending:Wait` state, and then again when the instance restarts and reaches the `Pending:Wait` state.
- When a lifecycle hook is configured for the `EC2_INSTANCE_TERMINATING` lifecycle action, a terminating instance pauses to perform a custom action when it reaches the `Terminating:Wait` state. However, if you specify an instance reuse policy to return instances to the warm pool on scale in instead of terminating them, then an instance that is returning to

the warm pool pauses to perform a custom action at the `Warmed:Pending:Wait` state for the `EC2_INSTANCE_TERMINATING` lifecycle action.

- If the demand on your application depletes the warm pool, Amazon EC2 Auto Scaling can launch instances directly into the Auto Scaling group as long as the group isn't at its maximum capacity yet. If the instances launch directly into the group, they are only paused to perform a custom action at the `Pending:Wait` state.
- To control how long an instance stays in a wait state before it transitions to the next state, configure your custom action to use the **complete-lifecycle-action** command. With lifecycle hooks, instances remain in a wait state either until you notify Amazon EC2 Auto Scaling that the specified lifecycle action is complete, or until the timeout period ends (one hour by default).

The following summarizes the flow for a scale-out event.



When instances reach a wait state, Amazon EC2 Auto Scaling sends a notification. Examples of these notifications are available in the EventBridge section of this guide. For more information, see Warm pool example events and patterns.

Supported notification targets

Amazon EC2 Auto Scaling provides support for defining any of the following as notification targets for lifecycle notifications:

- EventBridge rules

- Amazon SNS topics
- Amazon SQS queues

> **Important**
>
> Remember, if you have a user data (cloud-init) script in your launch template or launch configuration that configures your instances when they launch, you do not need to receive notifications to perform custom actions on instances that are starting or restarting.

The following sections contain links to documentation that describes how to configure notification targets:

EventBridge rules: To run code when Amazon EC2 Auto Scaling puts an instance into a wait state, you can create an EventBridge rule and specify a Lambda function as its target. To invoke different Lambda functions based on different lifecycle notifications, you can create multiple rules and associate each rule with a specific event pattern and Lambda function. For more information, see Create EventBridge rules for warm pool events.

Amazon SNS topics: To receive a notification when an instance is put into a wait state, you create an Amazon SNS topic and then set up Amazon SNS message filtering to deliver lifecycle notifications differently based on a message attribute. For more information, see Receive notifications using Amazon SNS.

Amazon SQS queues: To set up a delivery point for lifecycle notifications where a relevant consumer can pick them up and process them, you can create an Amazon SQS queue and a queue consumer that processes messages from the SQS queue. If you want the queue consumer to process lifecycle notifications differently based on a message attribute, you must also set up the queue consumer to parse the message and then act on the message when a specific attribute matches the desired value. For more information, see Receive notifications using Amazon SQS.

Create a warm pool for an Auto Scaling group

This topic describes how to create a warm pool for your Auto Scaling group.

> **Important**
>
> Before you continue, complete the prerequisites for creating a warm pool and confirm that you have created a lifecycle hook for your Auto Scaling group.

Create a warm pool

Use the following procedure to create a warm pool for your Auto Scaling group.

To create a warm pool (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up at the bottom of the page.
3. Choose the **Instance management** tab.
4. Under **Warm pool**, choose **Create warm pool**.
5. To configure a warm pool, do the following:

   a. For **Warm pool instance state**, choose which state you want to transition your instances to when they enter the warm pool. The default is `Stopped`.

   b. For **Minimum warm pool size**, enter the minimum number of instances to maintain in the warm pool.

   c. For **Instance reuse**, select the **Reuse on scale in** check box to allow instances in the Auto Scaling group to return to the warm pool on scale in.

   d. For **Warm pool size**, choose one of the available options:

      - **Default specification**: The size of the warm pool is determined by the difference between the maximum and desired capacity of the Auto Scaling group. This option streamlines warm pool management. After you create the warm pool, its size can be easily updated just by adjusting the maximum capacity of the group.
      - **Custom specification**: The size of the warm pool is determined by the difference between a custom value and the desired capacity of the Auto Scaling group. This option gives you flexibility to manage the size of your warm pool independently from the maximum capacity of the group.

6. View the **Estimated warm pool size based on current settings** section to confirm how the default or custom specification applies to the size of the warm pool. Remember, the warm pool size depends on the desired capacity of the Auto Scaling group, which will change if the group scales.
7. Choose **Create**.

Delete a warm pool

When you no longer need the warm pool, use the following procedure to delete it.

To delete your warm pool (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up at the bottom of the page.
3. Choose the **Instance management** tab.
4. For **Warm pool**, choose **Actions**, **Delete**.
5. When prompted for confirmation, choose **Delete**.

View health check status and the reason for health check failures

Health checks allow Amazon EC2 Auto Scaling to determine when an instance is unhealthy and should be terminated. For warm pool instances kept in a `Stopped` state, it employs the knowledge that Amazon EBS has of a `Stopped` instance's availability to identify unhealthy instances. It does this by calling the `DescribeVolumeStatus` API to determine the status of the EBS volume that's attached to the instance. For warm pool instances kept in a `Running` state, it relies on EC2 status checks to determine instance health. While there is no health check grace period for warm pool instances, Amazon EC2 Auto Scaling doesn't start checking instance health until the lifecycle hook finishes.

When an instance is found to be unhealthy, Amazon EC2 Auto Scaling automatically deletes the unhealthy instance and creates a new one to replace it. Instances are usually terminated within a few minutes after failing their health check. For more information, see View the reason for health check failures.

Custom health checks are also supported. This can be helpful if you have your own health check system that can detect an instance's health and send this information to Amazon EC2 Auto Scaling. For more information, see Set up a custom health check for your Auto Scaling group.

On the Amazon EC2 Auto Scaling console, you can view the status (healthy or unhealthy) of your warm pool instances. You can also view their health status using the AWS CLI or one of the SDKs.

To view the status of your warm pool instances (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Instance management** tab, under **Warm pool instances**, the **Lifecycle** column contains the state of your instances.

   The **Health status** column shows the assessment that Amazon EC2 Auto Scaling has made of instance health.

   > **Note**
   >
   > New instances start healthy. Until the lifecycle hook is finished, an instance's health is not checked.

To view the reason for health check failures (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched or terminated instances.

   If it terminated any unhealthy instances, the **Cause** column shows the date and time of the termination and the reason for the health check failure. For example, "At

2021-04-01T21:48:35Z an instance was taken out of service in response to EBS volume health check failure".

To view the status of your warm pool instances (AWS CLI)

View the warm pool for an Auto Scaling group by using the following describe-warm-pool command.

```
aws autoscaling describe-warm-pool --auto-scaling-group-name my-asg
```

Example output.

```
{
    "WarmPoolConfiguration": {
        "MinSize": 0,
        "PoolState": "Stopped"
    },
    "Instances": [
        {
            "InstanceId": "i-0b5e5e7521cfaa46c",
            "InstanceType": "t2.micro",
            "AvailabilityZone": "us-west-2a",
            "LifecycleState": "Warmed:Stopped",
            "HealthStatus": "Healthy",
            "LaunchTemplate": {
                "LaunchTemplateId": "lt-08c4cd42f320d5dcd",
                "LaunchTemplateName": "my-template-for-auto-scaling",
                "Version": "1"
            }
        },
        {
            "InstanceId": "i-0e21af9dcfb7aa6bf",
            "InstanceType": "t2.micro",
            "AvailabilityZone": "us-west-2a",
            "LifecycleState": "Warmed:Stopped",
            "HealthStatus": "Healthy",
            "LaunchTemplate": {
                "LaunchTemplateId": "lt-08c4cd42f320d5dcd",
                "LaunchTemplateName": "my-template-for-auto-scaling",
                "Version": "1"
            }
        }
```

```
    ]
}
```

To view the reason for health check failures (AWS CLI)

Use the following describe-scaling-activities command.

```
aws autoscaling describe-scaling-activities --auto-scaling-group-name my-asg
```

The following is an example response, where `Description` indicates that your Auto Scaling group has terminated an instance and `Cause` indicates the reason for the health check failure.

Scaling activities are ordered by start time. Activities still in progress are described first.

```
{
  "Activities": [
    {
      "ActivityId": "4c65e23d-a35a-4e7d-b6e4-2eaa8753dc12",
      "AutoScalingGroupName": "my-asg",
      "Description": "Terminating EC2 instance: i-04925c838b6438f14",
      "Cause": "At 2021-04-01T21:48:35Z an instance was taken out of service in
 response to EBS volume health check failure.",
      "StartTime": "2021-04-01T21:48:35.859Z",
      "EndTime": "2021-04-01T21:49:18Z",
      "StatusCode": "Successful",
      "Progress": 100,
      "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":\"us-west-2a
\"...}",
      "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:283179a2-
f3ce-423d-93f6-66bb518232f7:autoScalingGroupName/my-asg"
    },
...
  ]
}
```

Examples for creating and managing warm pools with the AWS CLI

You can create and manage warm pools using the AWS Management Console, AWS Command Line Interface (AWS CLI), or SDKs.

The following examples show you how to create and manage warm pools using the AWS CLI.

Contents

- Example 1: Keep instances in the Stopped state
- Example 2: Keep instances in the Running state
- Example 3: Keep instances in the Hibernated state
- Example 4: Return instances to the warm pool when scaling in
- Example 5: Specify the minimum number of instances in the warm pool
- Example 6: Define the warm pool size using a custom specification
- Example 7: Define an absolute warm pool size
- Example 8: Delete a warm pool

Example 1: Keep instances in the `Stopped` state

The following put-warm-pool example creates a warm pool that keeps instances in a `Stopped` state.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Stopped
```

Example 2: Keep instances in the `Running` state

The following put-warm-pool example creates a warm pool that keeps instances in a `Running` state instead of a `Stopped` state.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Running
```

Example 3: Keep instances in the `Hibernated` state

The following put-warm-pool example creates a warm pool that keeps instances in a `Hibernated` state instead of a `Stopped` state. This lets you stop instances without deleting their memory contents (RAM).

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Hibernated
```

Example 4: Return instances to the warm pool when scaling in

The following put-warm-pool example creates a warm pool that keeps instances in a `Stopped` state and includes the `--instance-reuse-policy` option. The instance reuse policy value `'{"ReuseOnScaleIn": true}'` tells Amazon EC2 Auto Scaling to return instances to the warm pool when your Auto Scaling group scales in.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Stopped --instance-reuse-policy '{"ReuseOnScaleIn": true}'
```

Example 5: Specify the minimum number of instances in the warm pool

The following put-warm-pool example creates a warm pool that maintains a minimum of 4 instances, so that there are at least 4 instances available to handle traffic spikes.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Stopped --min-size 4
```

Example 6: Define the warm pool size using a custom specification

By default, Amazon EC2 Auto Scaling manages the size of your warm pool as the difference between the maximum and desired capacity of the Auto Scaling group. However, you can manage the size of the warm pool independently from the group's maximum capacity by using the `--max-group-prepared-capacity` option.

The following put-warm-pool example creates a warm pool and sets the maximum number of instances that can exist concurrently in both the warm pool and Auto Scaling group. If the group has a desired capacity of 800, the warm pool will initially have a size of 100 as it initializes after running this command.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Stopped --max-group-prepared-capacity 900
```

To maintain a minimum number of instances in the warm pool, include the `--min-size` option with the command, as follows.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Stopped --max-group-prepared-capacity 900 --min-size 25
```

Example 7: Define an absolute warm pool size

If you set the same values for the `--max-group-prepared-capacity` and `--min-size` options, the warm pool has an absolute size. The following put-warm-pool example creates a warm pool that maintains a constant warm pool size of 10 instances.

```
aws autoscaling put-warm-pool --auto-scaling-group-name my-asg /
  --pool-state Stopped --min-size 10 --max-group-prepared-capacity 10
```

Example 8: Delete a warm pool

Use the following delete-warm-pool command to delete a warm pool.

```
aws autoscaling delete-warm-pool --auto-scaling-group-name my-asg
```

If there are instances in the warm pool, or if scaling activities are in progress, use the delete-warm-pool command with the `--force-delete` option. This option also terminates the Amazon EC2 instances and any outstanding lifecycle actions.

```
aws autoscaling delete-warm-pool --auto-scaling-group-name my-asg --force-delete
```

Detach or attach instances from your Auto Scaling group

You can detach instances from your Auto Scaling group. After an instance is detached, that instance becomes independent and can either be managed on its own or attached to a different Auto Scaling group, separate from the original group it belonged to. This can be useful, for example, when you want to perform testing using existing instances that are already running your application.

This topic provides instructions on how to detach and attach instances. When attaching instances, you can also use an existing instance rather than a detached one.

Instead of detaching and re-attaching an instance to the same group, we recommend using the standby procedure to temporarily remove the instance from the group. For more information, see Temporarily remove instances from your Auto Scaling group.

Contents

- Considerations for detaching instances

- Considerations for attaching instances
- Move an instance to a different group using detach and attach

Considerations for detaching instances

When you detach instances, keep these points in mind:

- You can detach an instance only when it's in the `InService` state.
- After you detach an instance, it continues running and incurring charges. To avoid unnecessary charges, make sure to reattach or terminate detached instances when they're no longer needed.
- You can choose to decrement the desired capacity by the number of instances that you are detaching. If you choose not to decrement the capacity, Amazon EC2 Auto Scaling launches new instances to replace the detached ones to maintain the desired capacity.
- If the number of instances that you are detaching will bring the Auto Scaling group below its minimum capacity, you must decrement the minimum capacity.
- If you detach multiple instances from the same Availability Zone without decrementing the desired capacity, the group will rebalance itself unless you suspend the `AZRebalance` process. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.
- If you detach an instance from an Auto Scaling group that has an attached load balancer target group or Classic Load Balancer, the instance is deregistered from the load balancer. If connection draining (deregistration delay) is enabled for your load balancer, Amazon EC2 Auto Scaling waits for in-flight requests to complete.

> **Note**
>
> If you are detaching instances that are in the `Standby` state, exercise caution. Attempting to detach instances after putting them into the `Standby` state may cause other instances to terminate unexpectedly.

Considerations for attaching instances

Note the following when attaching instances:

- Amazon EC2 Auto Scaling treats attached instances the same as instances launched by the group itself. This means that attached instances can be terminated during scale-in events if they're

selected. The permissions granted by the AWSServiceRoleForAutoScaling service-linked role allow Amazon EC2 Auto Scaling to do so.

- When you attach instances, the desired capacity of the group increases by the number of instances being attached. If the desired capacity after adding the new instances exceeds the maximum size of the group, the request to attach more instances fails.
- If you add instances to your group causing uneven distribution across Availability Zones, Amazon EC2 Auto Scaling rebalances the group to re-establish an even distribution unless you suspend the `AZRebalance` process. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.
- If you attach an instance to an Auto Scaling group that has an attached load balancer target group or Classic Load Balancer, the instance is registered with the load balancer.

For an instance to be attached, it must meet the following criteria:

- The instance is in the `running` state with Amazon EC2.
- The AMI used to launch the instance must still exist.
- The instance is not a member of another Auto Scaling group.
- The instance is launched into one of the Availability Zones defined in the Auto Scaling group.
- If the Auto Scaling group has an attached load balancer target group or Classic Load Balancer, the instance and the load balancer must both be in the same VPC.

Move an instance to a different group using detach and attach

Use one of the following procedures to detach an instance from your Auto Scaling group and attach it to a different Auto Scaling group.

To create a new Auto Scaling group from a detached instance, see Create an Auto Scaling group using parameters from an existing instance (not recommended, creates a launch configuration).

Console

To detach an instance from an Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

A split pane opens up in the bottom of the page.

3. On the **Instance management** tab, in **Instances**, select an instance and choose **Actions**, **Detach**.
4. In the **Detach instance** dialog box, keep the **Replace instance** check box selected to launch a replacement instance. Clear the check box to decrement the desired capacity.
5. When prompted for confirmation, type **`detach`** to confirm removing the specified instance from the Auto Scaling group, and then choose **Detach instance**.

You can now attach the instance to a different Auto Scaling group.

To attach an instance to an Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. (Optional) On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**. Select the Auto Scaling group and verify that the maximum size of the Auto Scaling group is large enough that you can add another instance. Otherwise, on the **Details** tab, increase the maximum capacity.
3. On the navigation pane, under **Instances**, choose **Instances**, and then select an instance.
4. Choose **Actions**, **Instance settings**, **Attach to Auto Scaling Group**.
5. On the **Attach to Auto Scaling group** page, for **Auto Scaling Group**, select the Auto Scaling group, and then choose **Attach**.
6. If the instance doesn't meet the criteria, you get an error message with the details. For example, the instance might not be in the same Availability Zone as the Auto Scaling group. Choose **Close** and try again with an Auto Scaling group that meets the criteria.

AWS CLI

To detach and attach an instance, use the following example commands. Replace each *`user input placeholder`* with your own information.

To detach an instance from an Auto Scaling group

1. To describe the current instances, use the following describe-auto-scaling-instances command.

```
aws autoscaling describe-auto-scaling-instances \
  --query 'AutoScalingInstances[?AutoScalingGroupName==`my-asg`]'
```

The following example shows the output produced when you run this command.

Take note of the ID of the instance that you intend to remove from the group. You need this ID in the next step.

```
{
    "AutoScalingInstances": [
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-05b4f7d5be44822a6",
            "InstanceType": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "InService"
        },
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-0c20ac468fa3049e8",
            "InstanceType": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "InService"
        },
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
```

```
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-0787762faf1c28619",
            "InstanceType": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "InService"
        },
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-0f280a4c58d319a8a",
            "InstanceType": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "InService"
        }
    ]
}
```

2. To detach an instance without decrementing the desired capacity, use the following detach-instances command.

```
aws autoscaling detach-instances --instance-ids i-05b4f7d5be44822a6 \
  --auto-scaling-group-name my-asg
```

To detach an instance and decrement the desired capacity, include the `--should-decrement-desired-capacity` option.

```
aws autoscaling detach-instances --instance-ids i-05b4f7d5be44822a6 \
  --auto-scaling-group-name my-asg --should-decrement-desired-capacity
```

You can now attach the instance to a different Auto Scaling group.

To attach an instance to an Auto Scaling group

1. To attach the instance to a different Auto Scaling group, use the following attach-instances command.

```
aws autoscaling attach-instances --instance-ids i-05b4f7d5be44822a6 --auto-
scaling-group-name my-asg-for-testing
```

2. To verify the size of the Auto Scaling group after attaching an instance, use the following describe-auto-scaling-groups command.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names my-asg-
for-testing
```

The following example response shows that the group has two running instances, one of which is the instance you attached.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg-for-testing",
            "AutoScalingGroupARN": "arn",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "2",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "MinSize": 1,
            "MaxSize": 5,
            "DesiredCapacity": 2,
            ...
            "Instances": [
                {
                    "ProtectedFromScaleIn": false,
                    "AvailabilityZone": "us-west-2a",
                    "LaunchTemplate": {
                        "LaunchTemplateName": "my-launch-template",
                        "Version": "1",
                        "LaunchTemplateId": "lt-050555ad16a3f9c7f"
                    },
                    "InstanceId": "i-05b4f7d5be44822a6",
                    "InstanceType": "t3.micro",
```

```
                    "HealthStatus": "Healthy",
                    "LifecycleState": "InService"
                },
                {
                    "ProtectedFromScaleIn": false,
                    "AvailabilityZone": "us-west-2a",
                    "LaunchTemplate": {
                        "LaunchTemplateName": "my-launch-template",
                        "Version": "2",
                        "LaunchTemplateId": "lt-050555ad16a3f9c7f"
                    },
                    "InstanceId": "i-00dcdfffdf5175890",
                    "InstanceType": "t3.micro",
                    "HealthStatus": "Healthy",
                    "LifecycleState": "InService"
                }
            ],
            ...
        }
    ]
}
```

Temporarily remove instances from your Auto Scaling group

You can put an instance that is in the `InService` state into the `Standby` state, update or troubleshoot the instance, and then return the instance to service. Instances that are on standby are still part of the Auto Scaling group, but they do not actively handle load balancer traffic.

This feature helps you stop and start the instances or reboot them without worrying about Amazon EC2 Auto Scaling terminating the instances as part of its health checks or during scale-in events.

For example, you can change the Amazon Machine Image (AMI) for an Auto Scaling group at any time by changing the launch template or launch configuration. Any subsequent instances that the Auto Scaling group launches use this AMI. However, the Auto Scaling group does not update the instances that are currently in service. You can terminate these instances and let Amazon EC2 Auto Scaling replace them, or use the instance refresh feature to terminate and replace the instances. Or, you can put the instances on standby, update the software, and then put the instances back in service.

Detaching instances from an Auto Scaling group is similar to putting instances on standby. Detaching instances might be useful if you want to attach them to a different group or manage the

instances like standalone EC2 instances and possibly terminate them. For more information, see Detach or attach instances from your Auto Scaling group.

Contents

- How the standby state works
- Considerations
- Health status of an instance in a standby state
- Temporarily remove an instance by setting it to standby

How the standby state works

The standby state works as follows to help you temporarily remove an instance from your Auto Scaling group:

1. You put an instance into the standby state. The instance remains in this state until you exit the standby state.
2. If there is a load balancer target group or Classic Load Balancer attached to your Auto Scaling group, the instance is deregistered from the load balancer. If connection draining is enabled for the load balancer, Elastic Load Balancing waits 300 seconds by default before completing the deregistration process, which helps in-flight requests to complete.
3. You can update or troubleshoot the instance.
4. You return the instance to service by exiting the standby state.
5. If there is a load balancer target group or Classic Load Balancer attached to your Auto Scaling group, the instance is registered with the load balancer.

For more information about the lifecycle of instances in an Auto Scaling group, see Amazon EC2 Auto Scaling instance lifecycle.

Considerations

The following are considerations when moving instances in and out of the standby state:

- When you put an instance on standby, you can either decrement the desired capacity through this operation, or keep it the same value.

  - If you choose not to decrement the desired capacity of the Auto Scaling group, Amazon EC2 Auto Scaling launches an instance to replace the one on standby. The intention is to help you maintain capacity for your application while one or more instances are on standby.
  - If you choose to decrement the desired capacity of the Auto Scaling group, this prevents the launch of an instance to replace the one on standby.
- After you put the instance back in service, the desired capacity is incremented to reflect how many instances are in the Auto Scaling group.
- To do the increment (and decrement), the new desired capacity must be between the minimum and maximum group size. Otherwise, the operation fails.
- If at anytime after putting an instance on standby, or returning the instance to service by exiting the standby state, your Auto Scaling group is found to not be balanced between Availability Zones, Amazon EC2 Auto Scaling compensates by rebalancing the Availability Zones unless you suspend the `AZRebalance` process. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.
- You are billed for instances that are in a standby state.

Health status of an instance in a standby state

Amazon EC2 Auto Scaling does not perform health checks on instances that are in a standby state. While the instance is in a standby state, its health status reflects the status that it had before you put it on standby. Amazon EC2 Auto Scaling does not perform a health check on the instance until you put it back in service.

For example, if you put a healthy instance on standby and then terminate it, Amazon EC2 Auto Scaling continues to report the instance as healthy. If you attempt to put a terminated instance that was on standby back in service, Amazon EC2 Auto Scaling performs a health check on the instance, determines that it is terminating and unhealthy, and launches a replacement instance. For more information, see Health checks for instances in an Auto Scaling group.

Temporarily remove an instance by setting it to standby

Use one of the following procedures to take an instance out of service temporarily by placing it into standby state.

Console

To temporarily remove an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Instance management** tab, in **Instances**, select an instance.
4. Choose **Actions**, **Set to Standby**.
5. In the **Set to Standby** dialog box, keep the **Replace instance** check box selected to launch a replacement instance. Clear the check box to decrement the desired capacity.
6. When prompted for confirmation, type **`standby`** to confirm putting the specified instance into the `Standby` state, and then choose **Set to Standby**.
7. You can update or troubleshoot your instance as needed. When you have finished, continue with the next step to return the instance to service.
8. Select the instance, choose **Actions**, **Set to InService**. In the **Set to InService** dialog box, choose **Set to InService**.

AWS CLI

To temporarily remove an instance from your Auto Scaling group, use the following example commands. Replace each *`user input placeholder`* with your own information.

To temporarily remove an instance

1. Use the following describe-auto-scaling-instances command to identify the instance to update.

   ```
   aws autoscaling describe-auto-scaling-instances \
     --query 'AutoScalingInstances[?AutoScalingGroupName==`my-asg`]'
   ```

   The following example shows the output produced when you run this command.

   Take note of the ID of the instance that you intend to remove from the group. You need this ID in the next step.

```
{
    "AutoScalingInstances": [
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-05b4f7d5be44822a6",
            "InstanceId": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "InService"
        },
       ...
    ]
}
```

2. Move the instance into a `Standby` state using the following enter-standby command. The `--should-decrement-desired-capacity` option decreases the desired capacity so that the Auto Scaling group does not launch a replacement instance.

```
aws autoscaling enter-standby --instance-ids i-05b4f7d5be44822a6 \
  --auto-scaling-group-name my-asg --should-decrement-desired-capacity
```

The following is an example response.

```
{
    "Activities": [
        {
            "ActivityId": "3b1839fe-24b0-40d9-80ae-bcd883c2be32",
            "AutoScalingGroupName": "my-asg",
            "Description": "Moving EC2 instance to Standby:
 i-05b4f7d5be44822a6",
            "Cause": "At 2023-12-15T21:31:26Z instance i-05b4f7d5be44822a6 was
 moved to standby
              in response to a user request, shrinking the capacity from 4 to
 3.",
            "StartTime": "2023-12-15T21:31:26.150Z",
```

```
            "StatusCode": "InProgress",
            "Progress": 50,
            "Details": "{\"Subnet ID\":\"subnet-c934b782\",\"Availability Zone
\":\"us-west-2a\"}"
        }
    ]
}
```

3. (Optional) Verify that the instance is in `Standby` using the following describe-auto-scaling-instances command.

```
aws autoscaling describe-auto-scaling-instances --instance-
ids i-05b4f7d5be44822a6
```

The following is an example response. Notice that the status of the instance is now `Standby`.

```
{
    "AutoScalingInstances": [
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-05b4f7d5be44822a6",
            "InstanceType": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "Standby"
        },
       ...
    ]
}
```

4. You can update or troubleshoot your instance as needed. When you have finished, continue with the next step to return the instance to service.

5. Put the instance back in service using the following exit-standby command.

```
aws autoscaling exit-standby --instance-ids i-05b4f7d5be44822a6 --auto-scaling-
group-name my-asg
```

The following is an example response.

```
{
    "Activities": [
        {
            "ActivityId": "db12b166-cdcc-4c54-8aac-08c5935f8389",
            "AutoScalingGroupName": "my-asg",
            "Description": "Moving EC2 instance out of Standby:
 i-05b4f7d5be44822a6",
            "Cause": "At 2023-12-15T21:46:14Z instance i-05b4f7d5be44822a6 was
 moved out of standby in
               response to a user request, increasing the capacity from 3 to
 4.",
            "StartTime": "2023-12-15T21:46:14.678Z",
            "StatusCode": "PreInService",
            "Progress": 30,
            "Details": "{\"Subnet ID\":\"subnet-c934b782\",\"Availability Zone
\":\"us-west-2a\"}"
        }
    ]
}
```

6. (Optional) Verify that the instance is back in service using the following `describe-auto-scaling-instances` command.

```
aws autoscaling describe-auto-scaling-instances --instance-
ids i-05b4f7d5be44822a6
```

The following is an example response. Notice that the status of the instance is `InService`.

```
{
    "AutoScalingInstances": [
        {
            "ProtectedFromScaleIn": false,
            "AvailabilityZone": "us-west-2a",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
```

```
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "InstanceId": "i-05b4f7d5be44822a6",
            "InstanceType": "t3.micro",
            "AutoScalingGroupName": "my-asg",
            "HealthStatus": "HEALTHY",
            "LifecycleState": "InService"
        },
        ...
    ]
}
```

Delete your Auto Scaling infrastructure

To completely delete your scaling infrastructure, complete the following tasks.

Tasks

- Delete your Auto Scaling group
- (Optional) Delete the launch configuration
- (Optional) Delete the launch template
- (Optional) Delete the load balancer and target groups
- (Optional) Delete CloudWatch alarms

Delete your Auto Scaling group

When you delete an Auto Scaling group, its desired, minimum, and maximum values are set to 0. As a result, the instances are terminated. Deleting an instance also deletes any associated logs or data, and any volumes on the instance. If you do not want to terminate one or more instances, you can detach them before you delete the Auto Scaling group. If the group has scaling policies, deleting the group deletes the policies, the underlying alarm actions, and any alarm that no longer has an associated action.

To delete your Auto Scaling group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group and choose **Actions**, **Delete**.

3. When prompted for confirmation, type **`delete`** to confirm deleting the specified Auto Scaling group and then choose **Delete**.

   A loading icon in the **Name** column indicates that the Auto Scaling group is being deleted. The **Desired**, **Min**, and **Max** columns show `0` instances for the Auto Scaling group. It takes a few minutes to terminate the instance and delete the group. Refresh the list to see the current state.

To delete your Auto Scaling group (AWS CLI)

Use the following delete-auto-scaling-group command to delete the Auto Scaling group. This operation does not work if the group has any EC2 instances; it is for group's with zero instances only.

```
aws autoscaling delete-auto-scaling-group --auto-scaling-group-name my-asg
```

If the group has instances or scaling activities in progress, use the delete-auto-scaling-group command with the `--force-delete` option. This will also terminate the EC2 instances. When you delete an Auto Scaling group from the Amazon EC2 Auto Scaling console, the console uses this operation to terminate any EC2 instances and delete the group at the same time.

```
aws autoscaling delete-auto-scaling-group --auto-scaling-group-name my-asg --force-
delete
```

(Optional) Delete the launch configuration

You can skip this step to keep the launch configuration for future use.

To delete the launch configuration (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the left navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Choose **Launch configurations** near the top of the page. When prompted for confirmation, choose **View launch configurations** to confirm that you want to view the **Launch configurations** page.
4. Select your launch configuration and choose **Actions**, **Delete launch configuration**.
5. When prompted for confirmation, choose **Delete**.

To delete the launch configuration (AWS CLI)

Use the following delete-launch-configuration command.

```
aws autoscaling delete-launch-configuration --launch-configuration-name my-launch-
config
```

(Optional) Delete the launch template

You can delete your launch template or just one version of your launch template. When you delete a launch template, all its versions are deleted.

You can skip this step to keep the launch template for future use.

To delete your launch template (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Instances**, choose **Launch Templates**.
3. Select your launch template and then do one of the following:
   - Choose **Actions**, **Delete template**. When prompted for confirmation, type **`Delete`** to confirm deleting the specified launch template and then choose **Delete**.
   - Choose **Actions**, **Delete template version**. Select the version to delete and choose **Delete**.

To delete the launch template (AWS CLI)

Use the following delete-launch-template command to delete your template and all its versions.

```
aws ec2 delete-launch-template --launch-template-id lt-068f72b72934aff71
```

Alternatively, you can use the delete-launch-template-versions command to delete a specific version of a launch template.

```
aws ec2 delete-launch-template-versions --launch-template-id lt-068f72b72934aff71 --
versions 1
```

(Optional) Delete the load balancer and target groups

Skip this step if your Auto Scaling group is not associated with an Elastic Load Balancing load balancer, or if you want to keep the load balancer for future use.

To delete your load balancer (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Load Balancing**, choose **Load Balancers**.
3. Choose the load balancer and choose **Actions**, **Delete**.
4. When prompted for confirmation, choose **Yes, Delete**.

To delete your target group (console)

1. On the navigation pane, under **Load Balancing**, choose **Target Groups**.
2. Choose the target group and choose **Actions**, **Delete**.
3. When prompted for confirmation, choose **Yes, Delete**.

To delete the load balancer associated with the Auto Scaling group (AWS CLI)

For Application Load Balancers and Network Load Balancers, use the following delete-load-balancer and delete-target-group commands.

```
aws elbv2 delete-load-balancer --load-balancer-arn my-load-balancer-arn
aws elbv2 delete-target-group --target-group-arn my-target-group-arn
```

For Classic Load Balancers, use the following delete-load-balancer command.

```
aws elb delete-load-balancer --load-balancer-name my-load-balancer
```

(Optional) Delete CloudWatch alarms

To delete the CloudWatch alarms associated with your Auto Scaling group, complete the following steps. For example, you might have alarms associated with step scaling or simple scaling policies.

> **Note**
>
> Deleting an Auto Scaling group automatically deletes the CloudWatch alarms that Amazon EC2 Auto Scaling manages for a target tracking scaling policy.

You can skip this step if your Auto Scaling group is not associated with any CloudWatch alarms, or if you want to keep the alarms for future use.

To delete the CloudWatch alarms (console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Alarms**.
3. Choose the alarms and choose **Action**, **Delete**.
4. When prompted for confirmation, choose **Delete**.

To delete the CloudWatch alarms (AWS CLI)

Use the delete-alarms command. You can delete one or more alarms at a time. For example, use the following command to delete the `Step-Scaling-AlarmHigh-AddCapacity` and `Step-Scaling-AlarmLow-RemoveCapacity` alarms.

```
aws cloudwatch delete-alarms --alarm-name Step-Scaling-AlarmHigh-AddCapacity Step-Scaling-AlarmLow-RemoveCapacity
```

Replace the instances in your Auto Scaling group

Amazon EC2 Auto Scaling offers capabilities that let you replace the Amazon EC2 instances in your Auto Scaling group after making updates, such as adding a new launch template with a new Amazon Machine Image (AMI) or adding new instance types. It also helps you streamline updates by giving you the option of including them in the same operation that replaces the instances.

This section includes information to help you do the following:

- Start an instance refresh to replace instances in your Auto Scaling group.
- Declare specific updates that describe a desired configuration and update the Auto Scaling group to the desired configuration.
- Skip replacing already updated instances.
- Use checkpoints to update instances in phases and perform verifications on your instances at specific points.
- Receive notifications by email when a checkpoint is reached.
- Use a rollback to restore the Auto Scaling group to the configuration it was previously using.
- Automatically roll back if the instance refresh fails for some reason or if any Amazon CloudWatch alarms you specify go into the `ALARM` state.
- Limit the lifetime of instances to provide consistent software versions and instance configurations across the Auto Scaling group.

Contents

- Use an instance refresh to update instances in an Auto Scaling group
- Replace Auto Scaling instances based on maximum instance lifetime

Use an instance refresh to update instances in an Auto Scaling group

You can use an instance refresh to update the instances in your Auto Scaling group. This feature can be useful when a configuration change requires you to replace instances, especially if your Auto Scaling group contains a large number of instances.

Some situations where an instance refresh can help include:

- Deploying a new Amazon Machine Image (AMI) or user data script across your Auto Scaling group. You can create a new launch template with the changes and then use an instance refresh to roll out the updates immediately.
- Migrating your instances to new instance types to take advantage of the latest improvements and optimizations.
- Switching your Auto Scaling groups from using a launch configuration to using a launch template. You can copy your launch configurations to launch templates and then use an instance refresh to update your instances to the new templates. For more information about migrating to launch templates, see Migrate your Auto Scaling groups to launch templates.

Contents

- How an instance refresh works in an Auto Scaling group
- Understand the default values for an instance refresh
- Start an instance refresh using the AWS Management Console or AWS CLI
- Monitor an instance refresh using the AWS Management Console or AWS CLI
- Cancel an instance refresh using the AWS Management Console or AWS CLI
- Undo changes with a manual or auto rollback
- Use an instance refresh with skip matching
- Add checkpoints to an instance refresh

How an instance refresh works in an Auto Scaling group

This topic describes how an instance refresh works and introduces the key concepts you need to understand to use it effectively.

Contents

- How it works
- Core concepts
- Health check grace period
- Instance type compatibility
- Limitations

How it works

To refresh instances in an Auto Scaling group, you can define a new configuration that contains the latest version of your application and any other updates you want to make. Then, start an instance refresh to replace existing instances with new ones based on that configuration.

To perform an instance refresh:

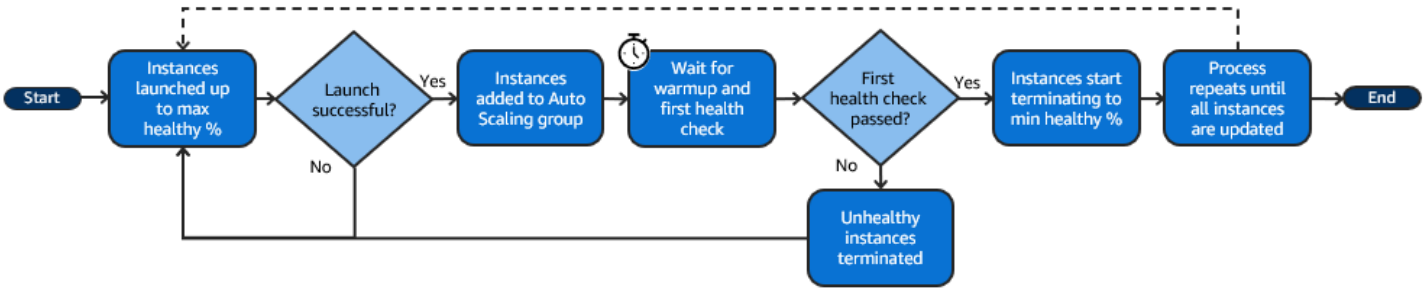
1. Create a new launch template or update the existing template with the desired configuration changes, such as a new Amazon Machine Image (AMI). For more information, see Create a launch template for an Auto Scaling group.
2. Start the instance refresh using the Amazon EC2 Auto Scaling console, AWS CLI, or SDK:
   - Specify the new launch template or launch template version you created. This will be used to launch new instances.
   - Set the preferred minimum and maximum healthy percentage. This controls how many instances are replaced simultaneously and whether new instances are launched before terminating old ones.
   - Configure any optional settings, such as:
     - **Checkpoints** – Pause the instance refresh after a certain percentage of replacements to verify progress.
     - **Skip matching** – Compare old instances to the new configuration and only replace those that don't match. When you start an instance refresh from the console, skip matching is on by default.
     - **Multiple instance types** – Apply a new or updated mixed instances policy as part of the desired configuration.

When the instance refresh has started, Amazon EC2 Auto Scaling will:

- Replace instances in batches based on the minimum and maximum healthy percentages.
- Launch the new instances first before terminating the old ones if the minimum healthy percentage is set to 100 percent. This ensures that your desired capacity is maintained at all times.
- Check instances for health status and give them time to warm up before more instances are replaced.
- Terminate and replace instances that are found to be unhealthy.

- Automatically update the Auto Scaling group settings with the new configuration changes after the instance refresh succeeds.
- Replace `InService` instances before instances that are in a warm pool.

The following flow diagram illustrates the launch before terminate behavior when you set the minimum healthy percentage to 100 percent.



> **Note**
>
> The minimum and maximum healthy percentages for an instance refresh only need to be specified if you have not set an instance maintenance policy, or if you need to override the existing policy. For more information, see Instance maintenance policies.
> Similarly, you only need to specify the instance warmup period for an instance refresh if you have not enabled the default warmup, or if you need to override the default. For more information, see Set the default instance warmup for an Auto Scaling group.

Core concepts

Before you get started, familiarize yourself with the following instance refresh core concepts:

Minimum healthy percentage

The *minimum healthy percentage* is the percentage of the desired capacity to keep in service, healthy, and ready to use during an instance refresh so that the refresh can continue. For example, if the minimum healthy percentage is 90 percent and the maximum healthy percentage is 100 percent, then 10 percent of capacity will be replaced at a time. If the new instances don't pass their health checks, Amazon EC2 Auto Scaling terminates and replaces them. If the instance refresh can't launch any healthy instances, it will eventually fail, leaving the other 90 percent of the group untouched. If the new instances stay healthy and finish their warmup period, Amazon EC2 Auto Scaling can continue to replace other instances.

An instance refresh can replace instances one at a time, several at a time, or all at once. To replace one instance at a time, set both the minimum and maximum healthy percentage to 100 percent. This changes the behavior of an instance refresh to launch before terminating, which prevents the group's capacity from falling below 100 percent of its desired capacity. To replace all instances at once, set a minimum healthy percentage of 0 percent.

Maximum healthy percentage

The *maximum healthy percentage* is the percentage of the desired capacity that your Auto Scaling group can increase to when replacing instances. The difference between the minimum and maximum cannot be greater than 100. A larger range increases the number of instances that can be replaced at the same time.

Instance warmup

The *instance warmup* is the time period from when a new instance's state changes to `InService` to when it is considered to have finished initializing. During an instance refresh, if the instances pass their health checks, Amazon EC2 Auto Scaling does not immediately move on to replacing the next instance after determining that a newly launched instance is healthy. It waits for the warmup period before it moves on to replacing the next instance. This can be helpful when your application still needs some initialization time before it responds to requests.

The instance warmup works the same way as the default instance warmup. Therefore, the same scaling considerations apply. For more information, see Set the default instance warmup for an Auto Scaling group.

Desired configuration

The *desired configuration* is the new configuration that you want Amazon EC2 Auto Scaling to deploy across your Auto Scaling group. For example, you can specify a new launch template and new instance types for your instances. During an instance refresh, Amazon EC2 Auto Scaling updates the Auto Scaling group to the desired configuration. If a scale-out event occurs during an instance refresh, Amazon EC2 Auto Scaling launches new instances with the desired configuration instead of the group's current settings. After the instance refresh succeeds, Amazon EC2 Auto Scaling updates the Auto Scaling group settings to reflect the new desired configuration that you specified as part of the instance refresh.

Skip matching

Skip matching tells Amazon EC2 Auto Scaling to ignore instances that already have your latest updates. This way, you don't replace more instances than you need to. This is helpful when you

want to make sure your Auto Scaling group uses a particular version of your launch template and only replaces those instances that use a different version.

Checkpoints

A *checkpoint* is a point in time where the instance refresh pauses for a specified amount of time. An instance refresh can contain multiple checkpoints. Amazon EC2 Auto Scaling emits events for each checkpoint. Therefore, you can add an EventBridge rule to send the events to a target, such as Amazon SNS, to be notified when a checkpoint is reached. After a checkpoint is reached, you have the opportunity to verify your deployment. If any problems are identified, you can cancel the instance refresh or roll it back. The ability to deploy updates in phases is a key benefit of checkpoints. If you don't use checkpoints, rolling replacements are performed continuously.

To learn more about all of the default settings you can configure when starting an instance refresh, see Understand the default values for an instance refresh.

Health check grace period

Amazon EC2 Auto Scaling determines whether an instance is healthy based on the status of the health checks that your Auto Scaling group uses. For more information, see Health checks for instances in an Auto Scaling group.

To make sure that these health checks start as soon as possible, don't set the group's health check grace period too high, but high enough for your Elastic Load Balancing health checks to determine whether a target is available to handle requests. For more information, see Set the health check grace period for an Auto Scaling group.

Instance type compatibility

Before you change your instance type, it's a good idea to verify that it works with your launch template. This confirms compatibility with the AMI that you specified. For example, let's say you launched your original instances from a paravirtual (PV) AMI, but you want to change to a current generation instance type that is only supported by a hardware virtual machine (HVM) AMI. In this case, you must use an HVM AMI in your launch template.

To confirm compatibility of the instance type without launching instances, use the run-instances command with the `--dry-run` option, as shown in the following example.

```
aws ec2 run-instances --launch-template LaunchTemplateName=my-template,Version='1' --
dry-run
```

For information about how compatibility is determined, see Compatibility for changing the instance type in the *Amazon EC2 User Guide*.

Limitations

- **Total duration**: The maximum amount of time that an instance refresh can continue to actively replace instances is 14 days.
- **Difference in behavior specific to weighted groups**: If a mixed instances group is configured with an instance weight that is larger than or equal to the group's desired capacity, Amazon EC2 Auto Scaling might replace all `InService` instances at once. To avoid this situation, follow the recommendation in the Configure an Auto Scaling group to use instance weights topic. Specify a desired capacity that is larger than your largest weight when you use weights with your Auto Scaling group.
- **One-hour timeout**: When an instance refresh is unable to continue making replacements because it is waiting to replace instances on standby or protected from scale in, or the new instances do not pass their health checks, Amazon EC2 Auto Scaling keeps retrying for an hour. It also provides a status message to help you resolve the issue. If the problem persists after an hour, the operation fails. The intention is to give it time to recover if there is a temporary issue.
- **Deploying code via user data**: Skip matching doesn't check for code changes that are deployed from a user data script. If you use user data to pull new code and install these updates on new instances, we recommend that you turn off skip matching to make sure that all instances receive your latest code, even without a launch template version update.
- **Update restriction**: If you attempt to update an Auto Scaling group's launch template, launch configuration, or mixed instances policy while an instance refresh with a desired configuration is active, the request will fail with the following validation error: `An active instance refresh with a desired configuration exists. All configuration options derived from the desired configuration are not available for update while the instance refresh is active.`

Understand the default values for an instance refresh

Before you start an instance refresh, you can customize various preferences that affect the instance refresh. Some preference defaults are different depending on whether you use the console or the command line (AWS CLI or AWS SDK).

The following table lists the default values for the instance refresh settings.

Setting	AWS CLI or AWS SDK	Amazon EC2 Auto Scaling console
CloudWatch alarm	Disabled (null)	Disabled
Auto rollback	Disabled (`false`)	Disabled
Checkpoints	Disabled (`false`)	Disabled
Checkpoint delay	1 hour (`3600` seconds)	1 hour
Instance warmup	The default instance warmup, if defined, or the health check grace period otherwise.	The default instance warmup, if defined, or the health check grace period otherwise.
Maximum healthy percentage	Varies based on your instance maintenance policy. If no instance maintenance policy, this defaults to 100 percent (null).	Varies based on your instance maintenance policy. If no instance maintenance policy, this defaults to 100 percent (null).
Minimum healthy percentage	Varies based on your instance maintenance policy. If no instance maintenance policy, this defaults to 90 percent.	Varies based on your instance maintenance policy. If no instance maintenance policy, this defaults to 90 percent.
Scale-in protected instances	Wait	Ignore
Skip matching	Disabled (`false`)	Enabled
Standby instances	Wait	Ignore

A description of each setting follows:

CloudWatch alarm (`AlarmSpecification`)

The CloudWatch alarm specification. CloudWatch alarms can be used to identify any issues and fail the operation if an alarm goes into the `ALARM` state. For more information, see Start an instance refresh with auto rollback.

Auto rollback (`AutoRollback`)

Controls whether Amazon EC2 Auto Scaling rolls back the Auto Scaling group to its previous configuration if the instance refresh fails. For more information, see Undo changes with a manual or auto rollback.

Checkpoints (`CheckpointPercentages`)

Controls whether Amazon EC2 Auto Scaling replaces instances in phases. This is useful if you need to perform verifications on your instances before replacing all instances. For more information, see Add checkpoints to an instance refresh.

Checkpoint delay (`CheckpointDelay`)

The amount of time, in seconds, to wait after a checkpoint before continuing. For more information, see Add checkpoints to an instance refresh.

Instance warmup (`InstanceWarmup`)

A time period, in seconds, during which Amazon EC2 Auto Scaling waits until a new instance is considered to have finished initializing before moving on to replacing the next instance. If you have already correctly defined a default instance warmup for the Auto Scaling group, then you don't need to change the instance warmup (unless you want to override the default). For more information, see Set the default instance warmup for an Auto Scaling group.

Maximum healthy percentage (`MaxHealthyPercentage`)

The percentage of the desired capacity of the Auto Scaling group that your group can increase to when replacing instances.

Minimum healthy percentage (`MinHealthyPercentage`)

The percentage of the desired capacity of the Auto Scaling group that must be in service, healthy, and ready to use before the operation can continue.

Scale-in protected instances (`ScaleInProtectedInstances`)

Controls what Amazon EC2 Auto Scaling does if instances that are protected from scale in are found. For more information about these instances, see Use instance scale-in protection to control instance termination.

Amazon EC2 Auto Scaling provides the following options:

- **Replace** (`Refresh`) – Replaces instances that are protected from scale in.
- **Ignore** (`Ignore`) – Ignores instances that are protected from scale in and continues to replace instances that are not protected.
- **Wait** (`Wait`) – Waits one hour for you to remove scale-in protection. If you don't do so, the instance refresh fails.

Skip matching (`SkipMatching`)

Controls whether Amazon EC2 Auto Scaling skips replacing instances that match the desired configuration. If no desired configuration is specified, then it skips replacing instances that have the same launch template and instance types that the Auto Scaling group was using before the instance refresh started. For more information, see Use an instance refresh with skip matching.

Standby instances (`StandbyInstances`)

Controls what Amazon EC2 Auto Scaling does if instances are found in `Standby` state. For more information about these instances, see Temporarily remove instances from your Auto Scaling group.

Amazon EC2 Auto Scaling provides the following options:

- **Terminate** (`Terminate`) – Terminates instances that are in `Standby`.
- **Ignore** (`Ignore`) – Ignores instances that are in `Standby` and continues to replace instances that are in the `InService` state.
- **Wait** (`Wait`) – Waits one hour for you to return the instances to service. If you don't do so, the instance refresh fails.

Start an instance refresh using the AWS Management Console or AWS CLI

> **Important**
>
> You can roll back an instance refresh that is in progress to undo any changes. For this to work, the Auto Scaling group must meet the prerequisites for using rollbacks before starting the instance refresh. For more information, see Undo changes with a manual or auto rollback.

The following procedures help you start an instance refresh using the AWS Management Console or AWS CLI.

Start an instance refresh (console)

If this is your first time starting an instance refresh, using the console will help you understand the features and options available.

Start an instance refresh in the console (basic procedure)

Use the following procedure if you have not previously defined a mixed instances policy for your Auto Scaling group. If you have previously defined a mixed instances policy, see Start an instance refresh in the console (mixed instances group) to start an instance refresh.

To start an instance refresh

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up at the bottom of the **Auto Scaling groups** page.
3. On the **Instance refresh** tab, in **Active instance refresh**, choose **Start instance refresh**.
4. For **Availability settings**, do the following:

   a. For **Instance replacement method**:

- If you *haven't* set an instance maintenance policy on the Auto Scaling group, the default setting for **Instance replacement method** is **Terminate and launch**. This is the legacy default behavior for an instance refresh.
- If you set an instance maintenance policy on the Auto Scaling group, it provides default values for **Instance replacement method**. To override the instance maintenance policy, choose **Override**. Overriding only applies to the current instance refresh. The next time you start an instance refresh, these values are reset to the instance maintenance policy defaults.

The following procedure explains how to update the instance replacement method.

i. Choose one of the following instance replacement methods:

   - **Launch before terminating**: A new instance must be provisioned first before an existing instance can be terminated. This is a good choice for applications that favor availability over cost savings.
   - **Terminate and launch**: New instances are provisioned at the same time your existing instances are terminated. This is a good choice for applications that favor cost savings over availability. It's also a good choice for applications that should not launch more capacity than is currently available.
   - **Custom behavior**: This option lets you set up a custom minimum and maximum range for the amount of capacity that you want available when replacing instances. This can help you achieve the right balance between cost and availability.

ii. For **Set healthy percentage**, enter values for one or both of the following fields. The enable fields vary depending on the option you choose for **Instance replacement method**.

   - **Min**: Sets the minimum healthy percentage that's required to proceed with the instance refresh.
   - **Max**: Sets the maximum healthy percentage that's possible during the instance refresh.

iii. Expand the **View estimated temporary capacity during replacements based on current group size** section to confirm how the values for **Min** and **Max** apply to your group. The exact values used depend on the desired capacity value, which will change if the group scales.

iv. Expand the **Set fallback behavior for invalid replacement sizes** section, and then choose whether to **Violate max healthy percentage** in order to prioritize availability, or **Violate min healthy percentage**.

Keeping the default **Violate min healthy percentage** option is not recommended for very small groups. If there is only one instance in the Auto Scaling group, starting an instance refresh can result in an outage.

This step configures fallback behavior if you are using an Auto Scaling group that doesn't have an instance maintenance policy yet. This option isn't available and doesn't appear when your group has an instance maintenance policy. This option is also only available for the **Terminate and launch** replacement method. Other replacement methods will violate the maximum healthy percentage in order to prioritize availability.

b. For **Instance warmup**, enter the number of seconds from when a new instance's state changes to `InService` to when it finishes initializing. Amazon EC2 Auto Scaling waits this amount of time before moving on to replace the next instance.

While warming up, a newly launched instance is also not counted toward the aggregated instance metrics of the Auto Scaling group (such as `CPUUtilization`, `NetworkIn`, and `NetworkOut`). If you added scaling policies to the Auto Scaling group, the scaling activities run in parallel. If you set a long interval for the instance refresh warmup period, it takes more time for newly launched instances to show up in the metrics. Therefore, an adequate warmup period keeps Amazon EC2 Auto Scaling from scaling on stale metric data.

If you have already correctly defined a default instance warmup for the Auto Scaling group, then you don't need to change the instance warmup. However, if you want to override the default, you can set a value for this option. For more information about setting the default instance warmup, see Set the default instance warmup for an Auto Scaling group.

5. For **Refresh settings**, do the following:

a. (Optional) For **Checkpoints**, choose **Enable checkpoints** to replace instances using an incremental or phased approach to an instance refresh. This provides additional time for verification between sets of replacements. If you choose not to enable checkpoints, the instances are replaced in one nearly continuous operation.

If you enable checkpoints, see Enable checkpoints (console) for additional steps.

b. Enable or turn off **Skip matching**:

- To skip replacing instances that already match your launch template, keep the **Enable skip matching** check box selected.
- If you turn off skip matching by clearing this check box, all instances can be replaced.

When you enable skip matching, you can set a new launch template or a new version of the launch template instead of using the existing one. Do this in the **Desired configuration** section of the **Start instance refresh** page.

> **Note**
>
> To use the skip matching feature to update an Auto Scaling group that currently uses a launch configuration, you must select a launch template in **Desired configuration**. Skip matching with a launch configuration is not supported.

c. For **Standby instances**, choose **Ignore**, **Terminate**, or **Wait**. This determines what happens if instances are found in `Standby` state. For more information, see Temporarily remove instances from your Auto Scaling group.

If you choose **Wait**, you must take additional steps to return these instances to service. If you don't, the instance refresh replaces all `InService` instances and waits for one hour. Then, if any `Standby` instances remain, the instance refresh fails. To prevent this situation, choose to **Ignore** or **Terminate** these instances instead.

d. For **Scale-in protected instances**, choose **Ignore**, **Replace**, or **Wait**. This determines what happens if scale-in protected instances are found. For more information, see Use instance scale-in protection to control instance termination.

If you choose **Wait**, you must take additional steps to remove scale-in protection from these instances. If you don't, the instance refresh replaces all unprotected instances and waits one hour. Then, if any scale-in protected instances remain, the instance refresh fails. To prevent this situation, choose to **Ignore** or **Replace** these instances instead.

6. (Optional) For **CloudWatch alarm**, choose **Enable CloudWatch alarms**, and then choose one or more alarms. CloudWatch alarms can be used to identify any issues and fail the operation if

an alarm goes into the `ALARM` state. For more information, see Start an instance refresh with auto rollback.

7. (Optional) Expand the **Desired configuration** section to specify updates that you want to make to your Auto Scaling group.

   For this step, you can choose to use JSON or YAML syntax to edit parameter values instead of making selections in the console interface. To do so, choose **Use code editor** instead of **Use console interface**. The following procedure explains how to make selections using the console interface.

   a. For **Update launch template**:

      - If you *haven't* created a new launch template or a new launch template version for your Auto Scaling group, don't select this check box.
      - If you created a new launch template or a new launch template version, select this check box. When you select this option, Amazon EC2 Auto Scaling shows you the current launch template and current launch template version. It also lists any other available versions. Choose the launch template and then choose the version.

        After you choose a version, you can see the version information. This is the version of the launch template that will be used when replacing instances as part of an instance refresh. If the instance refresh succeeds, this version of the launch template will also be used whenever new instances launch, such as when the group scales.

   b. For **Choose a set of instance types and purchase options to override the instance type in the launch template**:

      - Don't select this check box if you want to use the instance type and purchase option you specified in your launch template.
      - Select this check box if you want to override the instance type in the launch template or run Spot Instances. You can either manually add each instance type or choose a primary instance type and recommendation option that retrieves any additional matching instance types for you. If you plan to launch Spot Instances, we recommend adding a few different instance types. This way, Amazon EC2 Auto Scaling can launch another instance type if there is insufficient instance capacity in your chosen Availability Zones. For more information, see Auto Scaling groups with multiple instance types and purchase options.

> **Warning**
>
> Don't use Spot Instances with applications that can't handle a Spot Instance interruption. Interruptions can occur if the Amazon EC2 Spot service needs to reclaim capacity.

If you select this check box, make sure that the launch template doesn't already request Spot Instances. You can't use a launch template that requests Spot Instances to create an Auto Scaling group that uses multiple instance types and launches Spot and On-Demand Instances.

> **Note**
>
> To configure these options on an Auto Scaling group that currently uses a launch configuration, you must select a launch template in **Update launch template**. Overriding the instance type in your launch configuration is not supported.

8. (Optional) For **Rollback settings**, choose **Enable auto rollback** to automatically roll back the instance refresh if it fails.

   This setting can only be enabled when the Auto Scaling group meets the prerequisites for using rollbacks.

   For more information, see Undo changes with a manual or auto rollback.

9. Review all of your selections to confirm that everything is set up correctly.

   At this point, it's a good idea to verify that the differences between the current and proposed changes won't affect your application in unexpected or unwanted ways. To confirm that your instance type is compatible with your launch template, see Instance type compatibility.

10. When you are satisfied with your instance refresh selections, choose **Start instance refresh**.

Start an instance refresh in the console (mixed instances group)

Use the following procedure if you have created an Auto Scaling group with a mixed instances policy. If you haven't defined a mixed instances policy for your group yet, see Start an instance refresh in the console (basic procedure) to start an instance refresh.

To start an instance refresh

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up at the bottom of the **Auto Scaling groups** page.
3. On the **Instance refresh** tab, in **Active instance refresh**, choose **Start instance refresh**.
4. For **Availability settings**, do the following:

   a. For **Instance replacement method**:

      - If you *haven't* set an instance maintenance policy on the Auto Scaling group, the default setting for **Instance replacement method** is **Terminate and launch**. This is the legacy default behavior for an instance refresh.
      - If you set an instance maintenance policy on the Auto Scaling group, it provides default values for **Instance replacement method**. To override the instance maintenance policy, choose **Override**. Overriding only applies to the current instance refresh. The next time you start an instance refresh, these values are reset to the instance maintenance policy defaults.

      The following procedure explains how to update the instance replacement method.

      i. Choose one of the following instance replacement methods:

         - **Launch before terminating**: A new instance must be provisioned first before an existing instance can be terminated. This is a good choice for applications that favor availability over cost savings.
         - **Terminate and launch**: New instances are provisioned at the same time your existing instances are terminated. This is a good choice for applications that favor cost savings over availability. It's also a good choice for applications that should not launch more capacity than is currently available.
         - **Custom behavior**: This option lets you set up a custom minimum and maximum range for the amount of capacity that you want available when replacing instances. This can help you achieve the right balance between cost and availability.

ii. For **Set healthy percentage**, enter values for one or both of the following fields. The enable fields vary depending on the option you choose for **Instance replacement method**.

- **Min**: Sets the minimum healthy percentage that's required to proceed with the instance refresh.
- **Max**: Sets the maximum healthy percentage that's possible during the instance refresh.

iii. Expand the **View estimated temporary capacity during replacements based on current group size** section to confirm how the values for **Min** and **Max** apply to your group. The exact values used depend on the desired capacity value, which will change if the group scales.

iv. Expand the **Set fallback behavior for invalid replacement sizes** section, and then choose whether to **Violate max healthy percentage** in order to prioritize availability, or **Violate min healthy percentage**.

Keeping the default **Violate min healthy percentage** option is not recommended for very small groups. If there is only one instance in the Auto Scaling group, starting an instance refresh can result in an outage.

This step configures fallback behavior if you are using an Auto Scaling group that doesn't have an instance maintenance policy yet. This option isn't available and doesn't appear when your group has an instance maintenance policy. This option is also only available for the **Terminate and launch** replacement method. Other replacement methods will violate the maximum healthy percentage in order to prioritize availability.

b. For **Instance warmup**, enter the number of seconds from when a new instance's state changes to `InService` to when it finishes initializing. Amazon EC2 Auto Scaling waits this amount of time before moving on to replace the next instance.

While warming up, a newly launched instance is also not counted toward the aggregated instance metrics of the Auto Scaling group (such as `CPUUtilization`, `NetworkIn`, and `NetworkOut`). If you added scaling policies to the Auto Scaling group, the scaling activities run in parallel. If you set a long interval for the instance refresh warmup period, it takes more time for newly launched instances to show up in the metrics. Therefore, an adequate warmup period keeps Amazon EC2 Auto Scaling from scaling on stale metric data.

If you have already correctly defined a default instance warmup for the Auto Scaling group, then you don't need to change the instance warmup. However, if you want to override the default, you can set a value for this option. For more information about setting the default instance warmup, see Set the default instance warmup for an Auto Scaling group.

5. For **Refresh settings**, do the following:

   a. (Optional) For **Checkpoints**, choose **Enable checkpoints** to replace instances using an incremental or phased approach to an instance refresh. This provides additional time for verification between sets of replacements. If you choose not to enable checkpoints, the instances are replaced in one nearly continuous operation.

      If you enable checkpoints, see Enable checkpoints (console) for additional steps.

   b. Enable or turn off **Skip matching**:

      - To skip replacing instances that already match your launch template and any instance type overrides, keep the **Enable skip matching** check box selected.
      - If you choose to turn off skip matching by clearing this check box, all instances can be replaced.

      When you enable skip matching, you can set a new launch template or a new version of the launch template instead of using the existing one. Do this in the **Desired configuration** section of the **Start instance refresh** page. You can also update your instance type overrides in **Desired configuration**.

   c. For **Standby instances**, choose **Ignore**, **Terminate**, or **Wait**. This determines what happens if instances are found in `Standby` state. For more information, see Temporarily remove instances from your Auto Scaling group.

      If you choose **Wait**, you must take additional steps to return these instances to service. If you don't, the instance refresh replaces all `InService` instances and waits one hour. Then, if any `Standby` instances remain, the instance refresh fails. To prevent this situation, choose to **Ignore** or **Terminate** these instances instead.

   d. For **Scale-in protected instances**, choose **Ignore**, **Replace**, or **Wait**. This determines what happens if scale-in protected instances are found. For more information, see Use instance scale-in protection to control instance termination.

If you choose **Wait**, you must take additional steps to remove scale-in protection from these instances. If you don't, the instance refresh replaces all unprotected instances and waits one hour. Then, if any scale-in protected instances remain, the instance refresh fails. To prevent this situation, choose to **Ignore** or **Replace** these instances instead.

6. (Optional) For **CloudWatch alarm**, choose **Enable CloudWatch alarms**, and then choose one or more alarms. CloudWatch alarms can be used to identify any issues and fail the operation if an alarm goes into the `ALARM` state. For more information, see Start an instance refresh with auto rollback.

7. In the **Desired configuration** section, do the following.

   For this step, you can choose to use JSON or YAML syntax to edit parameter values instead of making selections in the console interface. To do so, choose **Use code editor** instead of **Use console interface**. The following procedure explains how to make selections using the console interface.

   a. For **Update launch template**:

      - If you *haven't* created a new launch template or a new launch template version for your Auto Scaling group, don't select this check box.
      - If you created a new launch template or a new launch template version, select this check box. When you select this option, Amazon EC2 Auto Scaling shows you the current launch template and current launch template version. It also lists any other available versions. Choose the launch template and then choose the version.

        After you choose a version, you can see the version information. This is the version of the launch template that will be used when replacing instances as part of an instance refresh. If the instance refresh succeeds, this version of the launch template will also be used whenever new instances launch, such as when the group scales.

   b. For **Use these settings to override the instance type and purchase option defined in the launch template**:

      By default, this check box is selected. Amazon EC2 Auto Scaling populates each parameter with the value that's currently set in the *mixed instances policy* for the Auto Scaling group. Only update values for the parameters that you want to change. For guidance on these settings, see Auto Scaling groups with multiple instance types and purchase options.

> **Warning**
>
> We recommend that you do not clear this check box. Only clear it if you want to stop using a mixed instances policy. After the instance refresh succeeds, Amazon EC2 Auto Scaling updates your group to match the **Desired configuration**. If it no longer includes a mixed instances policy, Amazon EC2 Auto Scaling gradually terminates any Spot Instances that are currently running and replaces them with On-Demand Instances. Or, if your launch template requests Spot Instances, then Amazon EC2 Auto Scaling gradually terminates any On-Demand Instances that are currently running and replaces them with Spot Instances.

8. (Optional) For **Rollback settings**, choose **Enable auto rollback** to automatically roll back the instance refresh if it fails for any reason.

   This setting can only be enabled when the Auto Scaling group meets the prerequisites for using rollbacks.

   For more information, see Undo changes with a manual or auto rollback.

9. Review all of your selections to confirm that everything is set up correctly.

   At this point, it's a good idea to verify that the differences between the current and proposed changes won't affect your application in unexpected or unwanted ways. To confirm that your instance type is compatible with your launch template, see Instance type compatibility.

   When you are satisfied with your instance refresh selections, choose **Start instance refresh**.

Start an instance refresh (AWS CLI)

To start an instance refresh

Use the following start-instance-refresh command to start an instance refresh from the AWS CLI. You can specify any preferences that you want to change in a JSON configuration file. When you reference the configuration file, provide the file path and name as shown in the following example.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`:

```
{
    "AutoScalingGroupName": "my-asg",
    "Preferences": {
      "InstanceWarmup": 60,
      "MinHealthyPercentage": 50,
      "AutoRollback": true,
      "ScaleInProtectedInstances": Ignore,
      "StandbyInstances": Terminate
    }
}
```

If preferences are not provided, default values are used. For more information, see Understand the default values for an instance refresh.

Example output:

```
{
    "InstanceRefreshId": "08b91cf7-8fa6-48af-b6a6-d227f40f1b9b"
}
```

Monitor an instance refresh using the AWS Management Console or AWS CLI

You can monitor an in progress instance refresh or look up the status of past instance refreshes from the last six weeks using the AWS Management Console or AWS CLI.

Monitor and check the status of an instance refresh

To monitor and check the status of an instance refresh, use one of the following methods:

Console

> **Tip**
>
> In this procedure, the named columns should already be displayed. To display hidden columns or change the number of rows shown, choose the gear icon on the top right corner of the section to open the preferences modal. Update the settings as needed and choose **Confirm**.

To monitor and check the status of an instance refresh (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up at the bottom of the page.
3. On the **Instance refresh** tab, under **Instance refresh history**, you can determine the status of your request by looking at the **Status** column. The operation goes into `Pending` status while it's initializing. The status should then quickly change to `InProgress`. When all instances are updated, the status changes to `Successful`.
4. You can further monitor the success or failure of in progress activities by viewing the group's scaling activities. On the **Activity** tab, under **Activity history**, when the instance refresh starts, you see entries when instances are terminated and another set of entries when instances are launched. If you have numerous scaling activities, you can see more of them by choosing the **>** icon at the top of the activity history. For information about troubleshooting issues that might cause activities to fail, see Troubleshoot issues in Amazon EC2 Auto Scaling.
5. (Optional) On the **Instance management** tab, under **Instances**, you can review the progress of specific instances as needed.

AWS CLI

To monitor and check the status of an instance refresh (AWS CLI)

Use the following describe-instance-refreshes command.

```
aws autoscaling describe-instance-refreshes --auto-scaling-group-name my-asg
```

The following is example output.

Instance refreshes are ordered by start time. Instance refreshes still in progress are described first.

```
{
  "InstanceRefreshes":[
    {
```

```
      "InstanceRefreshId":"08b91cf7-8fa6-48af-b6a6-d227f40f1b9b",
      "AutoScalingGroupName":"my-asg",
      "Status":"InProgress",
      "StatusReason":"Waiting for instances to warm up before continuing. For
 example: i-0645704820a8e83ff is warming up.",
      "StartTime":"2023-11-24T16:46:52+00:00",
      "PercentageComplete":50,
      "InstancesToUpdate":0,
      "Preferences":{
        "MaxHealthyPercentage":120,
        "MinHealthyPercentage":90,
        "InstanceWarmup":60,
        "SkipMatching":false,
        "AutoRollback":true,
        "ScaleInProtectedInstances":"Ignore",
        "StandbyInstances":"Ignore"
      }
    },
    {
      "InstanceRefreshId":"0e151305-1e57-4a32-a256-1fd14157c5ec",
      "AutoScalingGroupName":"my-asg",
      "Status":"Successful",
      "StartTime":"2023-11-22T13:53:37+00:00",
      "EndTime":"2023-11-22T13:59:45+00:00",
      "PercentageComplete":100,
      "InstancesToUpdate":0,
      "Preferences":{
        "MaxHealthyPercentage":120,
        "MinHealthyPercentage":90,
        "InstanceWarmup":60,
        "SkipMatching":false,
        "AutoRollback":true,
        "ScaleInProtectedInstances":"Ignore",
        "StandbyInstances":"Ignore"
      }
    }
  ]
}
```

You can further monitor the success or failure of in progress activities by viewing the group's scaling activities. The scaling activities also help you drill in for more details to help you troubleshoot issues with an instance refresh. For more information, see Troubleshoot issues in Amazon EC2 Auto Scaling.

Instance refresh statuses

When you start an instance refresh, it enters the **Pending** status. It passes from **Pending** to **InProgress** until it reaches **Successful**, **Failed**, **Cancelled**, **RollbackSuccessful**, or **RollbackFailed**.

An instance refresh can have the following statuses:

Status	Description
Pending	The request was created, but the instance refresh has not started.
InProgress	An instance refresh is in progress.
Successful	An instance refresh completed successfully.
Failed	An instance refresh failed to complete. You can troubleshoot using the status reason and the scaling activities.
Cancelling	An ongoing instance refresh is being cancelled.
Cancelled	The instance refresh is cancelled.
RollbackInProgress	An instance refresh is being rolled back.
RollbackFailed	The rollback failed to complete. You can troubleshoot using the status reason and the scaling activities.
RollbackSuccessful	The rollback completed successfully.

Cancel an instance refresh using the AWS Management Console or AWS CLI

You can cancel an instance refresh that is still in progress. You can't cancel it after it's finished.

Canceling an instance refresh does not roll back any instances that were already replaced. To roll back the changes to your instances, perform a rollback instead. For more information, see Undo changes with a manual or auto rollback.

Topics

- Cancel an instance refresh (console)

- Cancel an instance refresh (AWS CLI)

Cancel an instance refresh (console)

To cancel an instance refresh

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.
3. On the **Instance refresh** tab, in **Active instance refresh**, choose **Actions**, **Cancel**.
4. When prompted for confirmation, choose **Confirm**.

The status of the instance refresh is set to **Cancelling**. After the cancellation is complete, the status of the instance refresh is set to **Cancelled**.

Cancel an instance refresh (AWS CLI)

To cancel an instance refresh

Use the cancel-instance-refresh command from the AWS CLI and provide the Auto Scaling group name.

```
aws autoscaling cancel-instance-refresh --auto-scaling-group-name my-asg
```

Example output:

```
{
    "InstanceRefreshId": "08b91cf7-8fa6-48af-b6a6-d227f40f1b9b"
}
```

Undo changes with a manual or auto rollback

You can roll back an instance refresh that is still in progress. You can't roll it back after it's finished. You can, however, update your Auto Scaling group again by starting a new instance refresh.

When rolling back, Amazon EC2 Auto Scaling replaces the instances that have been deployed so far. The new instances match the configuration that you last saved on the Auto Scaling group before starting the instance refresh.

Amazon EC2 Auto Scaling provides the following ways to roll back:

- Manual rollback: You start a rollback manually to reverse what was deployed up to the rollback point.
- Auto rollback: Amazon EC2 Auto Scaling automatically reverses what was deployed if the instance refresh fails for some reason or if any CloudWatch alarms you specify go into the `ALARM` state.

Contents

- Considerations
- Manually start a rollback
- Start an instance refresh with auto rollback

Considerations

The following considerations apply when using a rollback:

- The rollback option is only available if you specify a desired configuration as part of starting an instance refresh.
- You can only roll back to a previous version of a launch template if the version is a specific numbered version. The rollback option is not available if the Auto Scaling group is configured to use the `$Latest` or `$Default` launch template version.
- You also cannot roll back to a launch template that is configured to use an AMI alias from the AWS Systems Manager Parameter Store.
- The configuration that you last saved on the Auto Scaling group must be in a stable state. If it's not in a stable state, the rollback workflow will still occur, but it will eventually fail. Until you resolve the issue, the Auto Scaling group might be in a failed state where it can no longer launch instances successfully. This might impact the availability of the service or application.

Manually start a rollback

Console

To manually start a rollback of an instance refresh (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.
3. On the **Instance refresh** tab, in **Active instance refresh**, choose **Actions**, **Start rollback**.
4. When prompted for confirmation, choose **Confirm**.

AWS CLI

To manually start a rollback of an instance refresh (AWS CLI)

Use the rollback-instance-refresh command from the AWS CLI and provide the Auto Scaling group name.

```
aws autoscaling rollback-instance-refresh --auto-scaling-group-name my-asg
```

Example output:

```
{
    "InstanceRefreshId": "08b91cf7-8fa6-48af-b6a6-d227f40f1b9b"
}
```

> **Tip**
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Start an instance refresh with auto rollback

Using the auto rollback feature, you can automatically roll back the instance refresh when it fails, such as when there are errors or a specified Amazon CloudWatch alarm goes into the `ALARM` state.

If you enable auto rollback, and there are errors while replacing instances, the instance refresh attempts to complete all replacements for one hour before it fails and rolls back. These errors are usually caused by things like EC2 launch failures, misconfigured health checks, or not ignoring or allowing the termination of instances that are in `Standby` state or protected from scale in.

Specifying CloudWatch alarms is optional. To specify an alarm, you first need to create it. You can specify metric alarms and composite alarms. For information about creating the alarm, see the Amazon CloudWatch User Guide. Using Elastic Load Balancing metrics as an example, if you use an Application Load Balancer, you could use the `HTTPCode_ELB_5XX_Count` and `HTTPCode_ELB_4XX_Count` metrics.

Considerations

- If you specify a CloudWatch alarm but do not enable auto rollback, and the alarm state goes to `ALARM`, the instance refresh fails without rolling back.
- You can choose a maximum of 10 alarms when you start an instance refresh.
- When choosing a CloudWatch alarm, the alarm must be in a compatible state. If the alarm state is `INSUFFICIENT_DATA` or `ALARM`, you receive an error when you try to start the instance refresh.
- When creating an alarm for Amazon EC2 Auto Scaling to use, the alarm should include how to treat missing data points. If a metric is frequently missing data points by design, the state of the alarm is `INSUFFICIENT_DATA` during those periods. When this happens, Amazon EC2 Auto Scaling cannot replace instances until new data points are found. To force the alarm to maintain the previous `ALARM` or `OK` state, you could choose to ignore missing data instead. For more information, see Configuring how alarms treat missing data in the *Amazon CloudWatch User Guide*.

Console

To start an instance refresh with auto rollback (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.
3. On the **Instance refresh** tab, in **Active instance refresh**, choose **Start instance refresh**.
4. Follow the Start an instance refresh (console) procedure and configure your instance refresh settings as needed.

5. (Optional) Under **Refresh settings**, for **CloudWatch alarm**, choose **Enable CloudWatch alarms**, and then choose one or more alarms to identify any issues and fail the operation if an alarm goes into the `ALARM` state.
6. Under **Rollback settings**, choose **Enable auto rollback** to automatically roll back a failed instance refresh to the configuration that you last saved on the Auto Scaling group before starting the instance refresh.
7. Review your selections, and then choose **Start instance refresh**.

AWS CLI

To start an instance refresh with auto rollback (AWS CLI)

Use the start-instance-refresh command and specify `true` for the `AutoRollback` option in the `Preferences`.

The following example shows how to start an instance refresh that will automatically roll back if something fails. Replace the *`italicized`* parameter values with your own.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`.

```
{
    "AutoScalingGroupName": "my-asg",
    "DesiredConfiguration": {
      "LaunchTemplate": {
          "LaunchTemplateName": "my-launch-template",
          "Version": "1"
       }
    },
    "Preferences": {
      "AutoRollback": true
    }
}
```

Alternatively, to automatically roll back when the instance refresh fails or when a specified CloudWatch alarm is in the `ALARM` state, specify the `AlarmSpecification` option in the `Preferences` and provide the alarm name, as in the following example. Replace the *`italicized`* parameter values with your own.

```
{
    "AutoScalingGroupName": "my-asg",
    "DesiredConfiguration": {
      "LaunchTemplate": {
          "LaunchTemplateName": "my-launch-template",
          "Version": "1"
       }
    },
    "Preferences": {
      "AutoRollback": true,
      "AlarmSpecification": { "Alarms": [ "my-alarm" ] }
    }
}
```

If successful, the command returns output similar to the following.

```
{
  "InstanceRefreshId": "08b91cf7-8fa6-48af-b6a6-d227f40f1b9b"
}
```

> **Tip**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Use an instance refresh with skip matching

Skip matching tells Amazon EC2 Auto Scaling to ignore instances that already have your latest updates. This way, you don't replace more instances than you need to. This is helpful when you want to make sure that your Auto Scaling group uses a particular version of your launch template and only replaces those instances that use a different version.

The following considerations apply to skip matching:

- If you start an instance refresh with both skip matching and a *desired configuration*, Amazon EC2 Auto Scaling checks to see if any instances match your desired configuration. Then, it only replaces the instances that don't match your desired configuration. After the instance refresh succeeds, Amazon EC2 Auto Scaling updates the group to reflect your desired configuration.

- If you start an instance refresh with skip matching, but you don't specify a desired configuration, Amazon EC2 Auto Scaling checks to see if any instances match the configuration that you last saved on the Auto Scaling group. Then, it only replaces the instances that don't match your last saved configuration.
- You can use skip matching with a new launch template, a new version of a launch template, or a set of instance types. If you enable skip matching, but none of these are changing, the instance refresh will succeed immediately without replacing any instances. If you made any other changes in your desired configuration (such as changing your Spot allocation strategy), Amazon EC2 Auto Scaling waits for the instance refresh to succeed. Then, it updates the Auto Scaling group settings to reflect the new desired configuration.
- You cannot use skip matching with a new launch configuration.
- When you start an instance refresh and provide a desired configuration, Amazon EC2 Auto Scaling ensures that all instances use your desired configuration. Therefore, when you specify either `$Default` or `$Latest` as the desired version for your launch template and then create a new version of the launch template while an instance refresh is in progress, any instances that were already replaced will be replaced again.
- Skip matching doesn't know whether a user data script in the launch template will pull updated code and install it on new instances. As a result, skip matching might skip replacing instances that have outdated code installed. In this case, you should turn off skip matching to make sure that all instances receive your latest code, even without a launch template version update.

This section includes AWS CLI instructions for starting an instance refresh with skip matching enabled. For instructions on using the console, see Start an instance refresh (console).

Skip matching (basic procedure)

Follow the steps in this section to use the AWS CLI to do the following:

- Create the launch template that you want to apply to your instances.
- Start an instance refresh to apply your launch template to your Auto Scaling group. If you do not enable skip matching, all instances will be replaced. This is true even if the launch template used to provision the instance is the same as the one that you specified for your desired configuration.

To use skip matching with a new launch template

1. Use the create-launch-template command to create a new launch template for your Auto Scaling group. Include the `--launch-template-data` option and JSON input that defines the details of the instances that are created for your Auto Scaling group.

   For example, use the following command to create a basic launch template with the AMI ID *ami-0123456789abcdef0* and the *t2.micro* instance type.

   ```
   aws ec2 create-launch-template --launch-template-name my-template-for-auto-scaling
    --version-description version1 \
     --launch-template-data
    '{"ImageId":"ami-0123456789abcdef0","InstanceType":"t2.micro"}'
   ```

   If successful, the command returns output similar to the following.

   ```
   {
      "LaunchTemplate": {
        "LaunchTemplateId": "lt-068f72b729example",
        "LaunchTemplateName": "my-template-for-auto-scaling",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "CreateTime": "2023-01-30T18:16:06.000Z",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
     }
   }
   ```

   For more information, see Examples for creating and managing launch templates with the AWS CLI.

2. Use the start-instance-refresh command to initiate the instance replacement workflow and apply your new launch template with the ID *lt-068f72b729example*. Because the launch template is new, it only has one version. This means that version 1 of the launch template is the target of this instance refresh. If a scale-out event occurs during the instance refresh, and Amazon EC2 Auto Scaling provisions new instances using the version 1 of this launch template, they will not be replaced. On successful completion of the operation, the new launch template is successfully applied to your Auto Scaling group.

   ```
   aws autoscaling start-instance-refresh --cli-input-json file://config.json
   ```

Contents of `config.json`.

```
{
    "AutoScalingGroupName": "my-asg",
    "DesiredConfiguration": {
      "LaunchTemplate": {
          "LaunchTemplateId": "lt-068f72b729example",
          "Version": "$Default"
       }
    },
    "Preferences": {
      "SkipMatching": true
    }
}
```

If successful, the command returns output similar to the following.

```
{
  "InstanceRefreshId": "08b91cf7-8fa6-48af-b6a6-d227f40f1b9b"
}
```

Skip matching (mixed instances group)

If you have an Auto Scaling group with a mixed instances policy, follow the steps in this section to use the AWS CLI to start an instance refresh with skip matching. You have the following options:

- Provide a new launch template to apply to all instance types specified in the policy.
- Provide an updated set of instance types with or without changing the launch template in the policy. For example, you might want to migrate away from unwanted instance types. You would use the launch template as is, without changing the AMI, security groups, or other specifics of the instances being replaced.

Follow the steps in one of the following sections, depending on which option fits your needs.

To use skip matching with a new launch template

1. Use the create-launch-template command to create a new launch template for your Auto Scaling group. Include the `--launch-template-data` option and JSON input that defines the details of the instances that are created for your Auto Scaling group.

   For example, use the following command to create a launch template with the AMI ID *ami-0123456789abcdef0*.

   ```
   aws ec2 create-launch-template --launch-template-name my-new-template --version-description version1 \
     --launch-template-data '{"ImageId":"ami-0123456789abcdef0"}'
   ```

   If successful, the command returns output similar to the following.

   ```
   {
      "LaunchTemplate": {
        "LaunchTemplateId": "lt-04d5cc9b88example",
        "LaunchTemplateName": "my-new-template",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "CreateTime": "2023-01-31T15:56:02.000Z",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
     }
   }
   ```

   For more information, see Examples for creating and managing launch templates with the AWS CLI.

2. To view the existing mixed instances policy for your Auto Scaling group, run the describe-auto-scaling-groups command. You'll need this information in the next step, when you start the instance refresh.

   The following example command returns the mixed instances policy configured for the Auto Scaling group named *my-asg*.

   ```
   aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
   ```

   If successful, the command returns output similar to the following.

```
{
  "AutoScalingGroups":[
    {
      "AutoScalingGroupName":"my-asg",
      "AutoScalingGroupARN":"arn",
      "MixedInstancesPolicy":{
        "LaunchTemplate":{
          "LaunchTemplateSpecification":{
            "LaunchTemplateId":"lt-073693ed27example",
            "LaunchTemplateName":"my-old-template",
            "Version":"$Default"
          },
          "Overrides":[
            {
              "InstanceType":"c5.large"
            },
            {
              "InstanceType":"c5a.large"
            },
            {
              "InstanceType":"m5.large"
            },
            {
              "InstanceType":"m5a.large"
            }
          ]
        },
        "InstancesDistribution":{
          "OnDemandAllocationStrategy":"prioritized",
          "OnDemandBaseCapacity":1,
          "OnDemandPercentageAboveBaseCapacity":50,
          "SpotAllocationStrategy":"price-capacity-optimized"
        }
      },
      "MinSize":1,
      "MaxSize":5,
      "DesiredCapacity":4,
      ...
    }
  ]
}
```

3. Use the start-instance-refresh command to initiate the instance replacement workflow and apply your new launch template with the ID *lt-04d5cc9b88example*. Because the launch template is new, it only has one version. This means that version 1 of the launch template is the target of this instance refresh. If a scale-out event occurs during the instance refresh, and Amazon EC2 Auto Scaling provisions new instances using the version 1 of this launch template, they will not be replaced. On successful completion of the operation, the updated mixed instances policy is successfully applied to your Auto Scaling group.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`.

```
{
  "AutoScalingGroupName":"my-asg",
  "DesiredConfiguration":{
    "MixedInstancesPolicy":{
      "LaunchTemplate":{
        "LaunchTemplateSpecification":{
          "LaunchTemplateId":"lt-04d5cc9b88example",
          "Version":"$Default"
        },
        "Overrides":[
          {
            "InstanceType":"c5.large"
          },
          {
            "InstanceType":"c5a.large"
          },
          {
            "InstanceType":"m5.large"
          },
          {
            "InstanceType":"m5a.large"
          }
        ]
      },
      "InstancesDistribution":{
        "OnDemandAllocationStrategy":"prioritized",
        "OnDemandBaseCapacity":1,
        "OnDemandPercentageAboveBaseCapacity":50,
        "SpotAllocationStrategy":"price-capacity-optimized"
```

```
        }
      }
    }
  },
  "Preferences":{
    "SkipMatching":true
  }
}
```

If successful, the command returns output similar to the following.

```
{
  "InstanceRefreshId": "08b91cf7-8fa6-48af-b6a6-d227f40f1b9b"
}
```

In this next procedure, you provide an updated set of instance types without changing the launch template.

To use skip matching with an updated set of instance types

1. To view the existing mixed instances policy for your Auto Scaling group, run the describe-auto-scaling-groups command. You'll need this information in the next step, when you start the instance refresh.

   The following example command returns the mixed instances policy configured for the Auto Scaling group named *my-asg*.

   ```
   aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
   ```

   If successful, the command returns output similar to the following.

   ```
   {
     "AutoScalingGroups":[
       {
         "AutoScalingGroupName":"my-asg",
         "AutoScalingGroupARN":"arn",
         "MixedInstancesPolicy":{
           "LaunchTemplate":{
             "LaunchTemplateSpecification":{
               "LaunchTemplateId":"lt-073693ed27example",
   ```

```
            "LaunchTemplateName":"my-template-for-auto-scaling",
            "Version":"$Default"
          },
          "Overrides":[
            {
              "InstanceType":"c5.large"
            },
            {
              "InstanceType":"c5a.large"
            },
            {
              "InstanceType":"m5.large"
            },
            {
              "InstanceType":"m5a.large"
            }
          ]
        },
        "InstancesDistribution":{
          "OnDemandAllocationStrategy":"prioritized",
          "OnDemandBaseCapacity":1,
          "OnDemandPercentageAboveBaseCapacity":50,
          "SpotAllocationStrategy":"price-capacity-optimized"
        }
      },
      "MinSize":1,
      "MaxSize":5,
      "DesiredCapacity":4,
      ...
    }
  ]
}
```

2. Use the start-instance-refresh command to initiate the instance replacement workflow and apply your updates. If you want to replace instances that use specific instance types, your desired configuration must specify the mixed instance policy with only the instance types that you want. You can choose whether to add new instance types in their place.

   The following example command starts an instance refresh without the unwanted instance type *`m5a.large`*. When an instance type in your group doesn't match one of the remaining three instance types, the instances are replaced. (Note that an instance refresh does not choose the instance types from which to provision the new instances; instead, the allocation

strategies do that.) On successful completion of the operation, the updated mixed instances policy is successfully applied to your Auto Scaling group.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`

```
{
  "AutoScalingGroupName":"my-asg",
  "DesiredConfiguration":{
    "MixedInstancesPolicy":{
      "LaunchTemplate":{
        "LaunchTemplateSpecification":{
          "LaunchTemplateId":"lt-073693ed27example",
          "Version":"$Default"
        },
        "Overrides":[
          {
            "InstanceType":"c5.large"
          },
          {
            "InstanceType":"c5a.large"
          },
          {
            "InstanceType":"m5.large"
          }
        ]
      },
      "InstancesDistribution":{
        "OnDemandAllocationStrategy":"prioritized",
        "OnDemandBaseCapacity":1,
        "OnDemandPercentageAboveBaseCapacity":50,
        "SpotAllocationStrategy":"price-capacity-optimized"
        }
      }
    }
  },
  "Preferences":{
    "SkipMatching":true
  }
}
```

Add checkpoints to an instance refresh

When using an instance refresh, you can choose to replace instances in phases, so that you can perform verifications on your instances as you go. To do a phased replacement, you add checkpoints, which are points in time where the instance refresh pauses. Using checkpoints gives you greater control over how you choose to update your Auto Scaling group. It helps you to confirm that your application will function in a reliable, predictable manner.

Contents

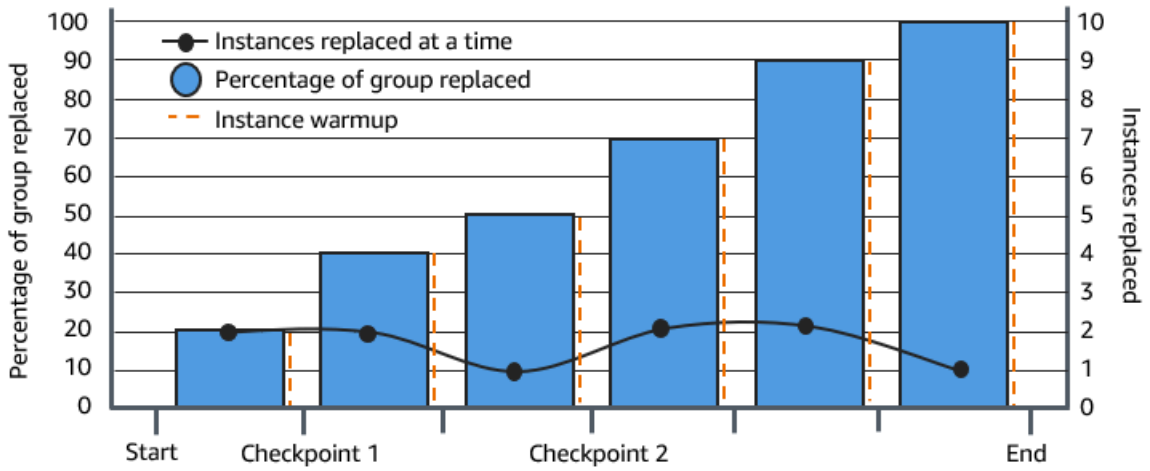
- How it works
- Considerations
- Enable checkpoints using the using the AWS Management Console or AWS CLI

How it works

When starting an instance refresh, you specify checkpoints as percentages of the total number of instances in the Auto Scaling group. These checkpoints indicate the minimum percentage of instances in the Auto Scaling group that must be new instances before the checkpoint is considered reached. For example, if your checkpoints are `[20, 50, 100]`, the first checkpoint is reached when 20 percent of instances are new, the second when 50 percent are new, and the final checkpoint when all instances are new.

Amazon EC2 Auto Scaling paces instance replacements to honor the specified checkpoint percentages while maintaining the group's minimum healthy percentage. To reach a checkpoint percentage, Amazon EC2 Auto Scaling will sometimes replace fewer but never more than what the minimum healthy percentage allows.

Consider the following Auto Scaling group that has 10 instances. The checkpoint percentages are `[20,50,100]`, the minimum healthy percentage is 80 percent, and the maximum healthy percentage is 100 percent. To maintain the minimum healthy percentage, only two instances can be replaced at a time. The following diagram summarizes the process for replacing instances before a checkpoint is reached.



In the above example, there is an instance warmup period for each new instance that starts. You might also have a lifecycle hook that puts an instance into a wait state and then performs a custom action as it's launching or terminating.

Amazon EC2 Auto Scaling emits events for each checkpoint except for the 100 percent complete checkpoint. You can add an EventBridge rule to send the events to a target such as Amazon SNS. This way, you are notified when you can run the required verifications. For more information, see Create EventBridge rules for instance refresh events.

Considerations

Keep the following considerations in mind when using checkpoints:

- Because checkpoints are based on percentages, the number of instances to replace changes with the size of the group. When a scale-out activity occurs and the size of the group increases, an in progress operation could reach a checkpoint again. If that happens, Amazon EC2 Auto Scaling sends another notification and repeats the wait time between checkpoints before continuing.
- It's possible to skip a checkpoint under certain circumstances. For example, suppose that your Auto Scaling group has two instances and your checkpoint percentages are `[10,40,100]`. After the first instance is replaced, Amazon EC2 Auto Scaling calculates that 50 percent of the group was replaced. Because 50 percent is higher than the first two checkpoints, it skips the first checkpoint (`10`) and sends a notification for the second checkpoint (`40`).
- Canceling the operation stops any further replacements from being made. If you cancel the operation or it fails before reaching the last checkpoint, any instances that were already replaced are not rolled back to their previous configuration.

- For a partial refresh, when you rerun the operation, Amazon EC2 Auto Scaling doesn't restart from the point of the last checkpoint, nor does it stop when only the earlier instances are replaced. However, it targets earlier instances for replacement first, before targeting new instances.
- The actual percentage complete might be higher than the percentage for that checkpoint when the checkpoint's percentage is too low relative to the number of instances in the group. For example, suppose the checkpoint's percentage is 20 percent and the group has four instances. If Amazon EC2 Auto Scaling replaces one of the four instances, the actual percentage replaced (25 percent) will be higher than the checkpoint's percentage (20 percent).
- After a checkpoint is reached, the displayed overall percentage complete doesn't update until after the instances finish warming up. For example, your checkpoint percentages are `[20,50]` with a checkpoint delay of 15 minutes and a minimum healthy percentage of 80 percent. Your Auto Scaling group has 10 instances and makes the following replacements:
  - 0:00: Two earlier instances are replaced with new ones.
  - 0:10: Two new instances finish warming up.
  - 0:25: Two earlier instances are replaced with new ones. (To maintain the minimum healthy percentage, only two instances are replaced.)
  - 0:35: Two new instances finish warming up.
  - 0:35: One earlier instance is replaced with a new one.
  - 0:45: One new instance finishes warming up.

  At 0:35, the operation stops launching new instances. The percentage complete doesn't accurately reflect the number of completed replacements yet (50 percent), because the new instance isn't done warming up. After the new instance completes its warmup period at 0:45, the percentage complete shows 50 percent.

Enable checkpoints using the using the AWS Management Console or AWS CLI

You can use the AWS Management Console or AWS CLI to enable checkpoints.

Enable checkpoints (console)

You can enable checkpoints before starting an instance refresh to replace instances using an incremental or phased approach. This provides additional time for verification.

To start an instance refresh that uses checkpoints

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up at the bottom of the **Auto Scaling groups** page.
3. On the **Instance refresh** tab, in **Active instance refresh**, choose **Start instance refresh**.
4. On the **Start instance refresh** page, enter the values for **Minimum healthy percentage** and **Instance warmup**.
5. Select the **Enable checkpoints** check box.

   This displays a box where you can define the percentage threshold for the first checkpoint.
6. For **Proceed until ____ % of the group is refreshed**, enter a number (1–100). This sets the percentage for the first checkpoint.
7. To add another checkpoint, choose **Add checkpoint** and then define the percentage for the next checkpoint.
8. To specify how long Amazon EC2 Auto Scaling waits after a checkpoint is reached, update the fields in **Wait for 1 hour between checkpoints**. The time unit can be hours, minutes, or seconds.
9. If you are finished with your instance refresh selections, choose **Start instance refresh**.

Enable checkpoints (AWS CLI)

To start an instance refresh with checkpoints enabled using the AWS CLI, you need a configuration file that defines the following parameters:

- `CheckpointPercentages`: Specifies threshold values for the percentage of instances to be replaced. These threshold values provide the checkpoints. When the percentage of instances that are replaced and warmed up reaches one of the specified thresholds, the operation waits for a specified period of time. You specify the number of seconds to wait in `CheckpointDelay`. When the specified period of time has passed, the instance refresh continues until it reaches the next checkpoint (if applicable).
- `CheckpointDelay`: Specifies the amount of time, in seconds, to wait after a checkpoint is reached before continuing. Choose a time period that provides enough time to perform your verifications.

The last value shown in the `CheckpointPercentages` array describes the percentage of the Auto Scaling group that needs to be successfully replaced. The operation transitions to `Successful` after this percentage is successfully replaced and each instance is considered to have finished initializing.

To create multiple checkpoints

To create multiple checkpoints, use the following example start-instance-refresh command. This example configures an instance refresh that initially refreshes one percent of the Auto Scaling group. After waiting 10 minutes, it then refreshes the next 19 percent and waits another 10 minutes. Finally, it refreshes the rest of the group before concluding the operation.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`:

```
{
    "AutoScalingGroupName": "my-asg",
    "Preferences": {
      "InstanceWarmup": 60,
      "MinHealthyPercentage": 80,
      "CheckpointPercentages": [1,20,100],
      "CheckpointDelay": 600
    }
}
```

To create a single checkpoint

To create a single checkpoint, use the following example start-instance-refresh command. This example configures an instance refresh that initially refreshes 20 percent of the Auto Scaling group. After waiting 10 minutes, it then refreshes the rest of the group before concluding the operation.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`:

```
{
    "AutoScalingGroupName": "my-asg",
    "Preferences": {
      "InstanceWarmup": 60,
```

```
      "MinHealthyPercentage": 80,
      "CheckpointPercentages": [20,100],
      "CheckpointDelay": 600
    }
}
```

To partially refresh the Auto Scaling group

To replace only a portion of your Auto Scaling group and then stop completely, use the following example start-instance-refresh command. This example configures an instance refresh that initially refreshes one percent of the Auto Scaling group. After waiting 10 minutes, it then refreshes the next 19 percent before concluding the operation.

```
aws autoscaling start-instance-refresh --cli-input-json file://config.json
```

Contents of `config.json`:

```
{
    "AutoScalingGroupName": "my-asg",
    "Preferences": {
      "InstanceWarmup": 60,
      "MinHealthyPercentage": 80,
      "CheckpointPercentages": [1,20],
      "CheckpointDelay": 600
    }
}
```

Replace Auto Scaling instances based on maximum instance lifetime

The maximum instance lifetime specifies the maximum amount of time (in seconds) that an instance can be in service before it is terminated and replaced. A common use case might be a requirement to replace your instances on a schedule because of internal security policies or external compliance controls.

You must specify a value of at least 86,400 seconds (one day). To clear a previously set value, specify a new value of 0. This setting applies to all current and future instances in your Auto Scaling group.

Contents

- Considerations
- Set the maximum instance lifetime
- Limitations

Considerations

The following are considerations when using this feature:

- Whenever an earlier instance is replaced and a new instance launches, the new instance uses the launch template or launch configuration that is currently associated with the Auto Scaling group. If your launch template or launch configuration specifies the Amazon Machine Image (AMI) ID of a different version of your application, this version of your application will be deployed automatically.
- Setting the maximum instance lifetime too low can cause instances to be replaced faster than desired. Amazon EC2 Auto Scaling will usually replace instances one at a time, with a pause between replacements. However, if the specified maximum instance lifetime doesn't provide enough time to replace each instance individually, Amazon EC2 Auto Scaling must replace more than one instance at a time. Several instances might be replaced at once, by up to 10 percent of the current capacity of your Auto Scaling group. To avoid replacing too many instances at once, either set a longer maximum instance lifetime or use instance scale-in protection to temporarily prevent individual instances from being terminated. For more information, see Use instance scale-in protection to control instance termination.
- By default, Amazon EC2 Auto Scaling creates a new scaling activity for terminating the instance and then terminates it. While the instance is terminating, another scaling activity launches a new instance. You can change this behavior to launch before terminating by using an instance maintenance policy. For more information, see Instance maintenance policies.

Set the maximum instance lifetime

When you create an Auto Scaling group in the console, you cannot set the maximum instance lifetime. However, after the group is created, you can edit it to set the maximum instance lifetime.

To set the maximum instance lifetime for a group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.

2. Select the check box next to the Auto Scaling group.

   A split pane opens up at the bottom of the **Auto Scaling groups** page, showing information about the group you selected.
3. On the **Details** tab, choose **Advanced configurations**, **Edit**.
4. For **Maximum instance lifetime**, enter the maximum number of seconds that an instance can be in service.
5. Choose **Update**.

On the **Activity** tab, under **Activity history**, you can view the replacement of instances in the group throughout its history.

To set the maximum instance lifetime for a group (AWS CLI)

You can also use the AWS CLI to set the maximum instance lifetime for new or existing Auto Scaling groups.

For new Auto Scaling groups, use the create-auto-scaling-group command.

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

The following is an example `config.json` file that shows a maximum instance lifetime of `2592000` seconds (30 days).

```
{
    "AutoScalingGroupName": "my-asg",
    "LaunchTemplate": {
        "LaunchTemplateName": "my-launch-template",
        "Version": "$Default"
    },
    "MinSize": 1,
    "MaxSize": 5,
    "MaxInstanceLifetime": 2592000,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782",
    "Tags": []
}
```

For existing Auto Scaling groups, use the update-auto-scaling-group command.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-existing-asg --
max-instance-lifetime 2592000
```

To verify the maximum instance lifetime for an Auto Scaling group

Use the describe-auto-scaling-groups command.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

Limitations

- **Maximum lifetime not guaranteed to be exact for every instance**: Instances are not guaranteed to be replaced only at the end of their maximum duration. In some situations, Amazon EC2 Auto Scaling might need to start replacing instances immediately after you update the maximum instance lifetime parameter. The reason for this behavior is to avoid replacing all instances at the same time.
- **Instance scale-in protection honored**: Amazon EC2 Auto Scaling provides instance scale-in protection to help you control which instances it can terminate. When this protection is enabled on an instance, Amazon EC2 Auto Scaling will not terminate the instance even if it has reached its maximum instance lifetime.
- **Instances terminated before launch**: When there is only one instance in the Auto Scaling group, the maximum instance lifetime feature can result in an outage because Amazon EC2 Auto Scaling terminates an instance and then launches a new instance by default. To change this behavior to launch before terminating, see Instance maintenance policies.

Increase or decrease compute capacity of your application with scaling

Scaling is the ability to increase or decrease the compute capacity of your application. Scaling starts with an event, or scaling action, which instructs an Auto Scaling group to either launch or terminate Amazon EC2 instances.

Amazon EC2 Auto Scaling provides a number of ways to adjust scaling to best meet the needs of your applications. As a result, it's important that you have a good understanding of your application. Keep the following considerations in mind:

- What role should Amazon EC2 Auto Scaling play in your application's architecture? It's common to think about automatic scaling primarily as a way to increase and decrease capacity, but it's also useful for maintaining a steady number of servers.
- What cost constraints are important to you? Because Amazon EC2 Auto Scaling uses EC2 instances, you pay only for the resources that you use. Knowing your cost constraints helps you decide when to scale your applications, and by how much.
- What metrics are important to your application? Amazon CloudWatch supports a number of different metrics that you can use with your Auto Scaling group.

Contents

- Choose your scaling method
- Set scaling limits for your Auto Scaling group
- Set the default instance warmup for an Auto Scaling group
- Manual scaling for Amazon EC2 Auto Scaling
- Scheduled scaling for Amazon EC2 Auto Scaling
- Dynamic scaling for Amazon EC2 Auto Scaling
- Predictive scaling for Amazon EC2 Auto Scaling
- Control which Auto Scaling instances terminate during scale in
- Suspend and resume Amazon EC2 Auto Scaling processes

Choose your scaling method

Amazon EC2 Auto Scaling provides several ways for you to scale your Auto Scaling group.

Maintain a fixed number of instances

The default for an Auto Scaling group is to not have any attached scaling policies or scheduled actions, which causes it to maintain a fixed size. After you create your Auto Scaling group, it starts by launching enough instances to meet its desired capacity. If there are no scaling conditions attached to the group, it continues to maintain its desired capacity even if an instance becomes unhealthy. Amazon EC2 Auto Scaling monitors the health of each instance in your Auto Scaling group. When it finds that an instance has become unhealthy, it replaces it with a new instance. You can read a more in-depth description of this process in Health checks for instances in an Auto Scaling group.

Scale manually

Manual scaling is the most basic way to scale your Auto Scaling group. You can either update the desired capacity of the Auto Scaling group or terminate instances in the Auto Scaling group. For more information, see Manual scaling for Amazon EC2 Auto Scaling.

Scale based on a schedule

Scaling by schedule means that scaling actions are performed automatically as a function of date and time. This is useful when you know exactly when to increase or decrease the number of instances in your group, simply because the need arises on a predictable schedule. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling.

Scale dynamically based on demand

A more advanced way to scale your resources, using dynamic scaling, lets you define a scaling policy that dynamically resizes your Auto Scaling group to meet changes in demand. For example, let's say that you have a web application that currently runs on two instances and you want the CPU utilization of the Auto Scaling group to stay at around 50 percent when the load on the application changes. This method is useful for scaling as traffic changes occur, when you don't know when the traffic will change. You can configure scaling policies to respond for you. There are multiple policy types (or a combination of them) that you can use to scale in response to traffic changes. For more information, see Dynamic scaling for Amazon EC2 Auto Scaling.

Scale proactively

You can also combine predictive scaling and dynamic scaling (proactive and reactive approaches, respectively) to scale your EC2 capacity faster. Use predictive scaling to increase the number of EC2 instances in your Auto Scaling group in advance of daily and weekly patterns in traffic flows. For more information, see Predictive scaling for Amazon EC2 Auto Scaling.

Set scaling limits for your Auto Scaling group

Scaling limits represent the minimum and maximum group size that you want for your Auto Scaling group. You set limits separately for the minimum and maximum size.

The group's desired capacity can be resized to a number that's within the range of your minimum and maximum size limits. The desired capacity must be equal to or greater than the minimum group size, and equal to or less than the maximum group size.

- **Desired capacity**: Represents the initial capacity of the Auto Scaling group at the time of creation. An Auto Scaling group attempts to maintain the desired capacity. It starts by launching the number of instances that are specified for the desired capacity, and maintains this number of instances as long as there are no scaling policies or scheduled actions attached to the Auto Scaling group.
- **Minimum capacity**: Represents the minimum group size. When scaling policies are set, they cannot decrease the group's desired capacity lower than the minimum capacity.
- **Maximum capacity**: Represents the maximum group size. When scaling policies are set, they cannot increase the group's desired capacity higher than the maximum capacity.

The minimum and maximum size limits also apply in the following scenarios:

- When you manually scale your Auto Scaling group by updating its desired capacity.
- When scheduled actions run that update the desired capacity. If a scheduled action runs without specifying new minimum and maximum size limits for the group, then the group's current minimum and maximum size limits apply.

An Auto Scaling group always tries to maintain its desired capacity. In cases where an instance terminates unexpectedly (for example, because of a Spot Instance interruption, a health check failure, or human action), the group automatically launches a new instance to maintain its desired capacity.

To manage these settings in the console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. On the **Auto Scaling groups** page, select the check box next to your Auto Scaling group.

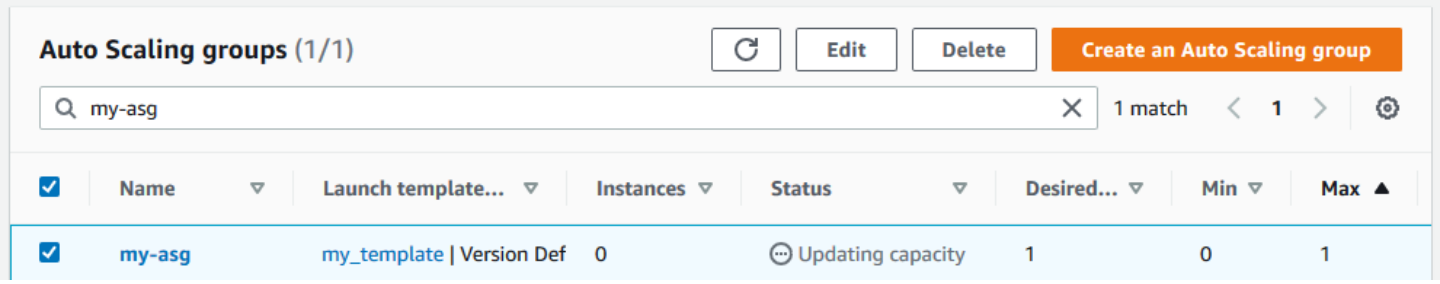
   A split pane opens up in the bottom of the page.
4. In the lower pane, in the **Details** tab, view or change the current settings for the group's desired, minimum, and maximum capacity. For more information, see Change the desired capacity of an existing Auto Scaling group.

Above the **Details** pane, you can find information such as the current number of instances in the Auto Scaling group, the desired, minimum, and maximum capacity, and a status column. If the Auto Scaling group uses instance weights, you can also find the number of capacity units contributed to the desired capacity.

To add or remove columns from the list, choose the settings icon at the top of the page. Then, for **Auto Scaling groups attributes**, turn each column on or off, and choose **Confirm**.

To verify the size of your Auto Scaling group after making changes

The **Instances** column shows the number of instances that are currently running. While an instance is being launched or terminated, the **Status** column displays a status of *Updating capacity*, as shown in the following image.



Wait for a few minutes, and then refresh the view to see the latest status. After a scaling activity completes, the **Instances** column shows an updated value.

You can view the number of instances and the status of the currently running instances from the **Instance management** tab, under **Instances**.

Set the default instance warmup for an Auto Scaling group

CloudWatch collects and aggregates usage data, such as CPU and network I/O, across your Auto Scaling instances. You use these metrics to create scaling policies that adjust the number of instances in your Auto Scaling group as the selected metric's value increases and decreases.

You can specify how long after an instance reaches the `InService` state it waits before contributing usage data to the aggregated metrics. This specified time is called the *default instance warmup*. This keeps dynamic scaling from being affected by metrics for individual instances that aren't yet handling application traffic and that might be experiencing temporarily high usage of compute resources.

To optimize the performance of your target tracking and step scaling policies, we strongly recommend that you enable and configure the default instance warmup. It is not enabled or configured by default.

When you enable the default instance warmup, keep in mind that if your Auto Scaling group is set to use an instance maintenance policy, or you use an instance refresh to replace instances, you can prevent instances from being counted toward the minimum healthy percentage before they have finished initializing.

Contents

- Scaling performance considerations
- Choose the default instance warmup time
- Enable the default instance warmup for a group
- Verify the default instance warmup time for a group
- Find scaling policies with a previously set instance warmup time
- Clear the previously set instance warmup for a scaling policy

Scaling performance considerations

It's useful for most applications to have one default instance warmup time that applies to all features, rather than different warmup times for different features. For example, if you don't set a default instance warmup, the instance refresh feature uses the health check grace period as the default warmup time. If you have any target tracking and step scaling policies, they use the value set for the default cooldown as the default warmup time. If you have any predictive scaling policies, they have no default warmup time.

While instances are warming up, your dynamic scaling policies scale out only if the metric value from instances that are not warming up is greater than the policy's alarm high threshold (or the target utilization of a target tracking scaling policy). If demand decreases, dynamic scaling becomes more conservative to protect your application's availability. This blocks the scale in activities for dynamic scaling until the new instances finish warming up.

While scaling out, Amazon EC2 Auto Scaling considers instances that are warming up as part of the capacity of the group when deciding how many instances to add to the group. Therefore, multiple alarm breaches that require a similar amount of capacity to be added result in a single scaling activity. The intention is to continuously scale out, without doing so excessively.

If default instance warmup is not enabled, the amount of time an instance waits before sending metrics to CloudWatch and counting it towards the current capacity will vary from instance to instance. So, there is the potential for your scaling policies to perform unpredictably compared to the actual workload that is occurring.

For example, consider an application with a recurring on-and-off workload pattern. A predictive scaling policy is used to make recurring decisions about whether to increase the number of instances. Because there is no default warmup time for predictive scaling policies, the instances start contributing to the aggregated metrics immediately. If these instances have higher resource usage on startup, then adding instances could cause the aggregated metrics to spike. Depending on how long it takes for usage to stabilize, this could impact any dynamic scaling policies that use these metrics. If a dynamic scaling policy's alarm high threshold is breached, then the group increases in size again. While the new instances are warming up, scale in activities will be blocked.

Choose the default instance warmup time

The key to setting the default instance warmup is determining how long your instances need to finish initializing and for resource consumption to stabilize after they reach the `InService` state. When choosing the instance warmup time, try to keep an optimal balance between collecting usage data for legitimate traffic, and minimizing data collection associated with temporary usage spikes on startup.

Suppose you have an Auto Scaling group attached to an Elastic Load Balancing load balancer. When new instances finish launching, they're registered to the load balancer before they enter the `InService` state. After the instances enter the `InService` state, resource consumption can still experience temporary spikes and need time to stabilize. For example, resource consumption for an application server that must download and cache large assets takes longer to stabilize than a

lightweight web server with no large assets to download. The instance warmup provides the time delay necessary for resource consumption to stabilize.

> **Important**
>
> If you're not sure how much time you need for the warmup time, you could start with 300 seconds. Then gradually decrease or increase it until you get the best scaling performance for your application. You might need to do this a few times to get it right. Alternatively, if you have any scaling policies that have their own warmup time (`EstimatedInstanceWarmup`), you could use this value to start. For more information, see Find scaling policies with a previously set instance warmup time.

Consider using lifecycle hooks for use cases where you have configuration tasks or scripts to run on startup. Lifecycle hooks can delay new instances from being put in service until they have finished initializing. They are particularly useful if you have bootstrapping scripts that take a while to complete. If you add a lifecycle hook, you can reduce the value of the default instance warmup. For more information about using lifecycle hooks, see Amazon EC2 Auto Scaling lifecycle hooks.

Enable the default instance warmup for a group

You can enable the default instance warmup when you create an Auto Scaling group. You can also enable it for existing groups.

By enabling the default instance warmup feature, you no longer have to specify values for warmup parameters for the following features:

- Instance refresh
- Target tracking scaling
- Step scaling

Console

To enable the default instance warmup for a new group (console)

When you create the Auto Scaling group, on the **Configure advanced options** page, under **Additional settings**, select the **Enable default instance warmup** option. Choose the warmup time that you need for your application.

AWS CLI

To enable the default instance warmup for a new group (AWS CLI)

To enable the default instance warmup for an Auto Scaling group, add the `--default-instance-warmup` option and specify a value, in seconds, from 0 to 3600. After it's enabled, a value of `-1` will turn this setting off.

The following create-auto-scaling-group command creates an Auto Scaling group with the name *my-asg* and enables the default instance warmup with a value of *120* seconds.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg --
default-instance-warmup 120 ...
```

> **Tip**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Console

To enable the default instance warmup for an existing group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your Auto Scaling group in.
3. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
4. On the **Details** tab, choose **Advanced configurations**, **Edit**.
5. For **Default instance warmup**, choose the warmup time that you need for your application.
6. Choose **Update**.

AWS CLI

To enable the default instance warmup for an existing group (AWS CLI)

The following example uses the update-auto-scaling-group command to enable the default instance warmup with a value of *120* seconds for an existing Auto Scaling group named *my-asg*.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg --
default-instance-warmup 120
```

> **Tip**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Verify the default instance warmup time for a group

Use the following procedure to verify the default instance warmup time for an Auto Scaling group using the AWS CLI.

To verify the default instance warmup time for an Auto Scaling group

Use the following describe-auto-scaling-groups command. Replace *my-asg* with the name of your Auto Scaling group.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

The following is an example response.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg",
            "AutoScalingGroupARN": "arn",
            ...
            "DefaultInstanceWarmup": 120
        }
    ]
}
```

Find scaling policies with a previously set instance warmup time

To identify whether you have policies that have their own warmup time for `EstimatedInstanceWarmup`, run the following describe-policies command using the AWS CLI. Replace *my-asg* with the name of your Auto Scaling group.

```
aws autoscaling describe-policies --auto-scaling-group-name my-asg
  --query 'ScalingPolicies[?EstimatedInstanceWarmup!=`null`]'
```

The following is example output.

```
[
  {
    "AutoScalingGroupName":"my-asg",
    "PolicyName":"cpu50-target-tracking-scaling-policy",
    "PolicyARN":"arn",
    "PolicyType":"TargetTrackingScaling",
    "StepAdjustments":[],
    "EstimatedInstanceWarmup":120,
    "Alarms":[{
        "AlarmARN": "arn:aws:cloudwatch:us-west-2:123456789012:alarm:TargetTracking-my-
asg-AlarmHigh-fc0e4183-23ac-497e-9992-691c9980c38e",
        "AlarmName": "TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e"
      },
      {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2",
            "AlarmName": "TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2"
    }],
    "TargetTrackingConfiguration":{
      "PredefinedMetricSpecification":{
        "PredefinedMetricType":"ASGAverageCPUUtilization"
      },
      "TargetValue":50.0,
      "DisableScaleIn":false
    },
    "Enabled":true
  },
```

```
    ... additional policies ...

]
```

Clear the previously set instance warmup for a scaling policy

After enabling the default instance warmup, update any scaling policies that have still their own warmup time to clear the previously set value. Otherwise, it will override the default instance warmup.

You can update scaling policies using the console, AWS CLI, or AWS SDKs. This section covers the steps for the console. If you use the AWS CLI or AWS SDKs, make sure you preserve the existing policy configuration, but remove the `EstimatedInstanceWarmup` property. When you update an existing scaling policy, the policy will be replaced with what you specify when you programmatically call PutScalingPolicy. The original values are not kept.

To clear the previously set instance warmup for a scaling policy (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Automatic scaling** tab, in **Dynamic scaling policies**, choose the policy you're interested in, and then choose **Actions**, **Edit**.
4. For **Instance warmup**, clear the instance warmup value to use the default instance warmup value instead.
5. Choose **Update**.

Manual scaling for Amazon EC2 Auto Scaling

You can manually adjust the number of EC2 instances in your Auto Scaling group at any time. This process of changing the instance count manually is referred to as *manual scaling*. Manual scaling is an alternative to auto scaling, especially if you want to make one-time capacity changes.

After you manually scale your group, Amazon EC2 Auto Scaling resumes normal auto scaling activities based on the scaling policies and scheduled actions that you defined. For groups with

default instance warmup enabled, any new instances go through a warmup period before they start contributing to the metrics used for auto scaling. This warmup period assists in stabilizing the group at the new capacity. For more information, see Set the default instance warmup for an Auto Scaling group.

Sometimes, you may want to temporarily disable scaling policies and scheduled actions before manually scaling a group. Doing so prevents conflicts from arising between manual scaling actions and automated scaling activities. For more information, see Turn off scaling activities.

Contents

- Change the desired capacity of an existing Auto Scaling group
- Terminate an instance in your Auto Scaling group (AWS CLI)

Change the desired capacity of an existing Auto Scaling group

When you change the desired capacity of your Auto Scaling group, Amazon EC2 Auto Scaling manages the process of launching and terminating instances to reach the new desired size.

Console

To change the size of your Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane displays at the bottom of the page.
3. On the **Details** tab, choose **Group details**, **Edit**.
4. For **Desired capacity**, increase or decrease the desired capacity. For example, to increase the size of the group by one, if the current value is 1, enter 2.

   If your new value for **Desired capacity** is greater than **Min desired capacity** and **Max desired capacity**, the **Max desired capacity** is automatically increased to the new desired capacity value.
5. When you are finished, choose **Update**.

Verify that the group size that you specified resulted in the same amount of instances being launched. For example, if you increased the size of the group by one, verify that your Auto Scaling group has launched one additional instance.

To verify that the size of your Auto Scaling group has changed

1. On the **Activity** tab, in **Activity history**, you can view the progress of activities that are associated with the Auto Scaling group. The **Status** column shows the current status of your instance. While your instance is launching, the status column shows `Not yet in service`. The status changes to `Successful` after the instance is launched. You can also use the refresh icon to see the current status of your instance. For more information, see Verify a scaling activity for an Auto Scaling group.
2. On the **Instance management** tab, in **Instances**, you can view the status of the instance. It takes a short time for an instance to launch.
   - The **Lifecycle** column shows the state of your instance. Initially, your instance is in the `Pending` state. After an instance is ready to receive traffic, its state is `InService`.
   - The **Health status** column shows the result of the Amazon EC2 Auto Scaling health checks on your instance.

AWS CLI

The following example assumes that you've created an Auto Scaling group with a minimum size of 1 and a maximum size of 5. Therefore, the group currently has one running instance.

To change the size of your Auto Scaling group

Use the set-desired-capacity command to change the size of your Auto Scaling group, as shown in the following example.

```
aws autoscaling set-desired-capacity --auto-scaling-group-name my-asg \
  --desired-capacity 2
```

If you choose to honor the default cooldown period for your Auto Scaling group, you must specify the `--honor-cooldown` option as shown in the following example. For more information, see Scaling cooldowns for Amazon EC2 Auto Scaling.

```
aws autoscaling set-desired-capacity --auto-scaling-group-name my-asg \
```

```
  --desired-capacity 2 --honor-cooldown
```

To verify the size of your Auto Scaling group

Use the describe-auto-scaling-groups command to confirm that the size of your Auto Scaling group has changed, as in the following example.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

The following is example output, which provides details about the group and instances launched.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg",
            "AutoScalingGroupARN": "arn",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-050555ad16a3f9c7f"
            },
            "MinSize": 1,
            "MaxSize": 5,
            "DesiredCapacity": 2,
            "DefaultCooldown": 300,
            "AvailabilityZones": [
                "us-west-2a"
            ],
            "LoadBalancerNames": [],
            "TargetGroupARNs": [],
            "HealthCheckType": "EC2",
            "HealthCheckGracePeriod": 300,
            "Instances": [
                {
                    "ProtectedFromScaleIn": false,
                    "AvailabilityZone": "us-west-2a",
                    "LaunchTemplate": {
                        "LaunchTemplateName": "my-launch-template",
                        "Version": "1",
                        "LaunchTemplateId": "lt-050555ad16a3f9c7f"
                    },
                    "InstanceId": "i-05b4f7d5be44822a6",
```

```
                    "InstanceType": "t3.micro",
                    "HealthStatus": "Healthy",
                    "LifecycleState": "Pending"
                },
                {
                    "ProtectedFromScaleIn": false,
                    "AvailabilityZone": "us-west-2a",
                    "LaunchTemplate": {
                        "LaunchTemplateName": "my-launch-template",
                        "Version": "1",
                        "LaunchTemplateId": "lt-050555ad16a3f9c7f"
                    },
                    "InstanceId": "i-0c20ac468fa3049e8",
                    "InstanceType": "t3.micro",
                    "HealthStatus": "Healthy",
                    "LifecycleState": "InService"
                }
            ],
            "CreatedTime": "2019-03-18T23:30:42.611Z",
            "SuspendedProcesses": [],
            "VPCZoneIdentifier": "subnet-c87f2be0",
            "EnabledMetrics": [],
            "Tags": [],
            "TerminationPolicies": [
                "Default"
            ],
            "NewInstancesProtectedFromScaleIn": false,
            "ServiceLinkedRoleARN": "arn",
            "TrafficSources": []
        }
    ]
}
```

Notice that `DesiredCapacity` shows the new value. Your Auto Scaling group has launched an additional instance.

Terminate an instance in your Auto Scaling group (AWS CLI)

There are times when you might want to manually scale in your Auto Scaling group but want to terminate a specific instance. You can manually scale in your Auto Scaling group by using the terminate-instance-in-auto-scaling-group command and specifying the ID of the instance you

want to terminate and the `--should-decrement-desired-capacity` option as shown in the following example.

```
aws autoscaling terminate-instance-in-auto-scaling-group \
  --instance-id i-026e4c9f62c3e448c --should-decrement-desired-capacity
```

The following is example output, which provides details about the scaling activity.

```
{
    "Activities": [
        {
            "ActivityId": "b8d62b03-10d8-9df4-7377-e464ab6bd0cb",
            "AutoScalingGroupName": "my-asg",
            "Description": "Terminating EC2 instance: i-026e4c9f62c3e448c",
            "Cause": "At 2023-09-23T06:39:59Z instance i-026e4c9f62c3e448c was taken
 out of service in response to a user request, shrinking the capacity from 1 to 0.",
            "StartTime": "2023-09-23T06:39:59.015000+00:00",
            "StatusCode": "InProgress",
            "Progress": 0,
            "Details": "{\"Subnet ID\":\"subnet-6194ea3b\",\"Availability Zone\":\"us-
west-2c\"}"
        }
    ]
}
```

This option is not available in the console. However, you can use the **Instances** page of the Amazon EC2 console to terminate an instance in your Auto Scaling group. When you do so, Amazon EC2 Auto Scaling detects that the instance is no longer running and replaces it automatically as part of the health check process. It takes a minute or two after you terminate the instance before a new instance launches. For information about how to terminate an instance, see Terminate an instance in the *Amazon EC2 User Guide*.

If you terminate instances in your group and that causes uneven distribution across Availability Zones, Amazon EC2 Auto Scaling rebalances the group to re-establish an even distribution unless you suspend the `AZRebalance` process. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.

Scheduled scaling for Amazon EC2 Auto Scaling

With scheduled scaling, you can set up automatic scaling for your application based on predictable load changes. You create scheduled actions that increase or decrease your group's desired capacity at specific times.

For example, you experience a regular weekly traffic pattern where load increases midweek and declines toward the end of the week. You can configure a scaling schedule in Amazon EC2 Auto Scaling that aligns with this pattern:

- On Wednesday morning, one scheduled action increases capacity by increasing the previously set desired capacity of the Auto Scaling group.
- On Friday evening, another scheduled action decreases capacity by decreasing the previously set desired capacity of the Auto Scaling group.

These scheduled scaling actions allow you to optimize costs and performance. Your application has sufficient capacity to handle the midweek traffic peak, but does not over-provision unneeded capacity at other times.

You can use scheduled scaling and scaling policies together to get the benefits of both approaches to scaling. After a scheduled scaling action runs, the scaling policy can continue to make decisions about whether to further scale capacity. This helps you ensure that you have sufficient capacity to handle the load for your application. While your application scales to match demand, current capacity must fall within the minimum and maximum capacity that was set by your scheduled action.

Contents

- How scheduled scaling works
- Recurring schedules
- Time zone
- Considerations
- Limitations
- Create a scheduled action
- View scheduled action details
- Delete a scheduled action

How scheduled scaling works

To use scheduled scaling, create *scheduled actions*, which tell Amazon EC2 Auto Scaling to perform scaling activities at specific times. When you create a scheduled action, you specify the Auto Scaling group, when the scaling activity should occur, the new desired capacity, and optionally a new minimum capacity and a new maximum capacity. You can create scheduled actions that scale one time only or that scale on a recurring schedule.

At the specified time, Amazon EC2 Auto Scaling scales based on the new capacity values, by comparing current capacity to the specified desired capacity.

- If current capacity is less than the specified desired capacity, Amazon EC2 Auto Scaling scales out, or adds instances, to the specified desired capacity.
- If current capacity is greater than the specified desired capacity, Amazon EC2 Auto Scaling scales in, or removes instances, to the specified desired capacity.

A scheduled action sets the group's desired, minimum, and maximum capacity at the date and time specified. You can create a scheduled action for only one of these capacities at a time, for example, desired capacity. However, there are some cases where you must include the minimum and maximum capacity to ensure that the desired capacity that you specified in the action is not outside of these limits.

Recurring schedules

To create a recurring schedule using the AWS CLI or an SDK, specify a cron expression and a time zone to describe when that scheduled action is to recur. You can optionally specify a date and time for the start time, the end time, or both.

To create a recurring schedule using the AWS Management Console, specify the recurrence pattern, time zone, start time, and optional end time of your scheduled action. All of the recurrence pattern options are based on cron expressions. Alternatively, you can write your own custom cron expression.

The supported cron expression format consists of five fields separated by white spaces: [Minute] [Hour] [Day_of_Month] [Month_of_Year] [Day_of_Week]. For example, the cron expression `30 6 * * 2` configures a scheduled action that recurs every Tuesday at 6:30 AM. The asterisk is used as a wildcard to match all values for a field. For other examples of cron expressions, see https://

crontab.guru/examples.html. For information about writing your own cron expressions in this format, see Crontab.

Choose your start and end times carefully. Keep the following in mind:

- If you specify a start time, Amazon EC2 Auto Scaling performs the action at this time, and then performs the action based on the specified recurrence.
- If you specify an end time, the action stops repeating after this time. A scheduled action does not persist in your account once it has reached its end time.
- If a recurrence time exactly matches the end time, Amazon EC2 Auto Scaling will not perform the scheduled action at the end time.
- The start time and end time must be set in UTC when you use the AWS CLI or an SDK.

Time zone

By default, the recurring schedules that you set are in Coordinated Universal Time (UTC). You can change the time zone to correspond to your local time zone or a time zone for another part of your network. When you specify a time zone that observes Daylight Saving Time (DST), the action automatically adjusts for DST.

The valid values are the canonical names for time zones from the Internet Assigned Numbers Authority (IANA) Time Zone database. For example, US Eastern time is canonically identified as `America/New_York`. For more information, see https://www.iana.org/time-zones.

Location-based time zones such as `America/New_York` automatically adjust for DST. However, a UTC-based time zone such as `Etc/UTC` is an absolute time and will not adjust for DST.

For example, you have a recurring schedule whose time zone is `America/New_York`. The first scaling action happens in the `America/New_York` time zone before DST starts. The next scaling action happens in the `America/New_York` time zone after DST starts. The first action starts at 8:00 AM UTC-5 in local time, while the second time starts at 8:00 AM UTC-4 in local time.

If you create a scheduled action using the AWS Management Console and specify a time zone that observes DST, both the recurring schedule and the start and end times automatically adjust for DST.

Considerations

When you create a scheduled action, keep the following in mind:

- The order of execution for scheduled actions is guaranteed within the same group, but not for scheduled actions across groups.
- A scheduled action generally runs within seconds. However, the action might be delayed for up to two minutes from the scheduled start time. Because scheduled actions within an Auto Scaling group are executed in the order that they are specified, actions with scheduled start times close to each other can take longer to execute.
- You can temporarily turn off scheduled scaling for an Auto Scaling group by suspending the `ScheduledActions` process. This helps you prevent scheduled actions from being active without having to delete them. You can then resume scheduled scaling when you want to use it again. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.
- After creating a scheduled action, you can update any of its settings except the name.

Limitations

- The names of scheduled actions must be unique per Auto Scaling group.
- A scheduled action must have a unique time value. If you attempt to schedule an activity at a time when another scaling activity is already scheduled, the call is rejected and returns an error indicating that a scheduled action with this scheduled start time already exists.
- You can create a maximum of 125 scheduled actions per Auto Scaling group.

Create a scheduled action

To create a scheduled action for your Auto Scaling group, use one of the following methods:

Console

To create a scheduled action

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Automatic scaling** tab, in **Scheduled actions**, choose **Create scheduled action**.
4. Enter a **Name** for the scheduled action.

5. For **Desired capacity**, **Min**, **Max**, choose the new desired capacity of the group and the new minimum and maximum size limits. The desired capacity must be equal to or greater than the minimum group size, and equal to or less than the maximum group size.
6. For **Recurrence**, choose one of the available options.
   - If you want to scale on a recurring schedule, choose how often Amazon EC2 Auto Scaling should run the scheduled action.
     - If you choose an option that begins with **Every**, the cron expression is created for you.
     - If you choose **Cron**, enter a cron expression that specifies when to perform the action.
   - If you want to scale only once, choose **Once**.
7. For **Time zone**, choose a time zone. The default is `Etc/UTC`.

   All of the time zones listed are from the IANA Time Zone database. For more information, see https://en.wikipedia.org/wiki/List_of_tz_database_time_zones.
8. Define a date and time for **Specific start time**.
   - If you chose a recurring schedule, the start time defines when the first scheduled action in the recurring series runs.
   - If you chose **Once** as the recurrence, the start time defines the date and time for the schedule action to run.
9. (Optional) For recurring schedules, you can specify an end time by choosing **Set End Time** and then choosing a date and time for **End by**.
10. Choose **Create**. The console displays the scheduled actions for the Auto Scaling group.

AWS CLI

To create a scheduled action, you can use one of the following example commands. Replace each *`user input placeholder`* with your own information.

Example: To scale one time only

Use the following put-scheduled-update-group-action command with the `--start-time "YYYY-MM-DDThh:mm:ssZ"` and `--desired-capacity` options.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-one-time-action \
```

```
  --auto-scaling-group-name my-asg --start-time "2021-03-31T08:00:00Z" --desired-
capacity 3
```

Example: To schedule scaling on a recurring schedule

Use the following put-scheduled-update-group-action command with the `--recurrence "cron expression"` and `--desired-capacity` options.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-
recurring-action \
  --auto-scaling-group-name my-asg --recurrence "0 9 * * *" --desired-capacity 3
```

By default, Amazon EC2 Auto Scaling runs the specified recurrence schedule based on the UTC time zone. To specify a different time zone, include the `--time-zone` option and the name of the IANA time zone, as in the following example.

```
--time-zone "America/New_York"
```

For more information, see https://en.wikipedia.org/wiki/List_of_tz_database_time_zones.

View scheduled action details

To view details of upcoming scheduled actions for your Auto Scaling group, use one of the following methods:

Console

To view scheduled action details

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select your Auto Scaling group.
3. On the **Automatic scaling** tab, in the **Scheduled actions** section, you can view upcoming scheduled actions.

Note that the console shows the values for **Start time** and **End time** in your local time with the UTC offset in effect at the specified date and time. The UTC offset is the difference, in hours and

minutes, from local time to UTC. The value for **Time zone** shows your requested time zone, for example, `America/New_York`.

AWS CLI

Use the following describe-scheduled-actions command.

```
aws autoscaling describe-scheduled-actions --auto-scaling-group-name my-asg
```

If successful, this command returns output similar to the following.

```
{
  "ScheduledUpdateGroupActions": [
    {
      "AutoScalingGroupName": "my-asg",
      "ScheduledActionName": "my-recurring-action",
      "Recurrence": "30 0 1 1,6,12 *",
      "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-action",
      "StartTime": "2020-12-01T00:30:00Z",
      "Time": "2020-12-01T00:30:00Z",
      "MinSize": 1,
      "MaxSize": 6,
      "DesiredCapacity": 4
    }
  ]
}
```

Verify scaling activities

To verify the scaling activities associated with scheduled scaling, see Verify a scaling activity for an Auto Scaling group.

Delete a scheduled action

To delete a scheduled action, use one of the following methods:

Console

To delete a scheduled action

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select your Auto Scaling group.
3. On the **Automatic scaling** tab, in **Scheduled actions**, select a scheduled action.
4. Choose **Actions**, **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

AWS CLI

Use the following delete-scheduled-action command.

```
aws autoscaling delete-scheduled-action --auto-scaling-group-name my-asg \
  --scheduled-action-name my-recurring-action
```

Dynamic scaling for Amazon EC2 Auto Scaling

Dynamic scaling scales the capacity of your Auto Scaling group as traffic changes occur.

Amazon EC2 Auto Scaling supports the following types of dynamic scaling policies:

- **Target tracking scaling**—Increase and decrease the current capacity of the group based on a Amazon CloudWatch metric and a target value. It works similar to the way that your thermostat maintains the temperature of your home—you select a temperature and the thermostat does the rest.
- **Step scaling**—Increase and decrease the current capacity of the group based on a set of scaling adjustments, known as *step adjustments*, that vary based on the size of the alarm breach.
- **Simple scaling**—Increase and decrease the current capacity of the group based on a single scaling adjustment, with a cooldown period between each scaling activity.

We strongly recommend that you use target tracking scaling policies and choose a metric that changes inversely proportional to a change in the capacity of your Auto Scaling group. So if you double the size of your Auto Scaling group, the metric decreases by 50 percent. This allows the

metric data to accurately trigger proportional scaling events. Included are metrics like average CPU utilization or average request count per target.

With target tracking, your Auto Scaling group scales in direct proportion to the actual load on your application. That means that in addition to meeting the immediate need for capacity in response to load changes, a target tracking policy can also adapt to load changes that take place over time, for example, due to seasonal variations.

Target tracking policies also remove the need to manually define CloudWatch alarms and scaling adjustments. Amazon EC2 Auto Scaling handles this automatically based on the target you set.

Contents

- How dynamic scaling policies work
- Multiple dynamic scaling policies
- Target tracking scaling policies for Amazon EC2 Auto Scaling
- Step and simple scaling policies for Amazon EC2 Auto Scaling
- Scaling cooldowns for Amazon EC2 Auto Scaling
- Scaling policy based on Amazon SQS
- Verify a scaling activity for an Auto Scaling group
- Disable a scaling policy for an Auto Scaling group
- Delete a scaling policy for an Auto Scaling group
- Example scaling policies for the AWS CLI

How dynamic scaling policies work

A dynamic scaling policy instructs Amazon EC2 Auto Scaling to track a specific CloudWatch metric, and it defines what action to take when the associated CloudWatch alarm is in ALARM. The metrics that are used to invoke the alarm state are an aggregation of metrics coming from all of the instances in the Auto Scaling group. (For example, let's say you have an Auto Scaling group with two instances where one instance is at 60 percent CPU and the other is at 40 percent CPU. On average, they are at 50 percent CPU.) When the policy is in effect, Amazon EC2 Auto Scaling adjusts the group's desired capacity up or down when the threshold of an alarm is breached.

When a dynamic scaling policy is invoked, if the capacity calculation produces a number outside of the minimum and maximum size range of the group, Amazon EC2 Auto Scaling ensures that the new capacity never goes outside of the minimum and maximum size limits. Capacity is measured in

one of two ways: using the same units that you chose when you set the desired capacity in terms of instances, or using capacity units (if instance weights are applied).

- Example 1: An Auto Scaling group has a maximum capacity of 3, a current capacity of 2, and a dynamic scaling policy that adds 3 instances. When invoking this policy, Amazon EC2 Auto Scaling adds only 1 instance to the group to prevent the group from exceeding its maximum size.
- Example 2: An Auto Scaling group has a minimum capacity of 2, a current capacity of 3, and a dynamic scaling policy that removes 2 instances. When invoking this policy, Amazon EC2 Auto Scaling removes only 1 instance from the group to prevent the group from becoming less than its minimum size.

When the desired capacity reaches the maximum size limit, scaling out stops. If demand drops and capacity decreases, Amazon EC2 Auto Scaling can scale out again.

The exception is when you use instance weights. In this case, Amazon EC2 Auto Scaling can scale out above the maximum size limit, but only by up to your maximum instance weight. Its intention is to get as close to the new desired capacity as possible but still adhere to the allocation strategies that are specified for the group. The allocation strategies determine which instance types to launch. The weights determine how many capacity units each instance contributes to the desired capacity of the group based on its instance type.

- Example 3: An Auto Scaling group has a maximum capacity of 12, a current capacity of 10, and a dynamic scaling policy that adds 5 capacity units. Instance types have one of three weights assigned: 1, 4, or 6. When invoking the policy, Amazon EC2 Auto Scaling chooses to launch an instance type with a weight of 6 based on the allocation strategy. The result of this scale-out event is a group with a desired capacity of 12 and a current capacity of 16.

Multiple dynamic scaling policies

In most cases, a target tracking scaling policy is sufficient to configure your Auto Scaling group to scale out and scale in automatically. A target tracking scaling policy allows you to select a desired outcome and have the Auto Scaling group add and remove instances as needed to achieve that outcome.

For an advanced scaling configuration, your Auto Scaling group can have more than one scaling policy. For example, you can define one or more target tracking scaling policies, one or more step scaling policies, or both. This provides greater flexibility to cover multiple scenarios.

To illustrate how multiple dynamic scaling policies work together, consider an application that uses an Auto Scaling group and an Amazon SQS queue to send requests to a single EC2 instance. To help ensure that the application performs at optimum levels, there are two policies that control when the Auto Scaling group should scale out. One is a target tracking policy that uses a custom metric to add and remove capacity based on the number of SQS messages in the queue. The other is a step scaling policy that uses the Amazon CloudWatch `CPUUtilization` metric to add capacity when the instance exceeds 90 percent utilization for a specified length of time.

When there are multiple policies in force at the same time, there's a chance that each policy could instruct the Auto Scaling group to scale out (or in) at the same time. For example, it's possible that the `CPUUtilization` metric spikes and breaches the threshold of the CloudWatch alarm at the same time that the SQS custom metric spikes and breaches the threshold of the custom metric alarm.

When these situations occur, Amazon EC2 Auto Scaling chooses the policy that provides the largest capacity for both scale out and scale in. Suppose, for example, that the policy for `CPUUtilization` launches one instance, while the policy for the SQS queue launches two instances. If the scale-out criteria for both policies are met at the same time, Amazon EC2 Auto Scaling gives precedence to the SQS queue policy. This results in the Auto Scaling group launching two instances.

The approach of giving precedence to the policy that provides the largest capacity applies even when the policies use different criteria for scaling in. For example, if one policy terminates three instances, another policy decreases the number of instances by 25 percent, and the group has eight instances at the time of scale in, Amazon EC2 Auto Scaling gives precedence to the policy that provides the largest number of instances for the group. This results in the Auto Scaling group terminating two instances (25 percent of 8 = 2). The intention is to prevent Amazon EC2 Auto Scaling from removing too many instances.

We recommend caution, however, when using target tracking scaling policies with step scaling policies because conflicts between these policies can cause undesirable behavior. For example, if the step scaling policy initiates a scale in activity before the target tracking policy is ready to scale in, the scale in activity will not be blocked. After the scale in activity completes, the target tracking policy could instruct the group to scale out again.

Target tracking scaling policies for Amazon EC2 Auto Scaling

A target tracking scaling policy automatically scales the capacity of your Auto Scaling group based on a target metric value. This allows your application to maintain optimal performance and cost efficiency without manual intervention.

With target tracking, you select a metric and a target value to represent the ideal average utilization or throughput level for your application. Amazon EC2 Auto Scaling creates and manages the CloudWatch alarms that invoke scaling events when the metric deviates from the target. As an example, this is similar to how a thermostat maintains a target temperature.

For example, let's say that you currently have an application that runs on two instances, and you want the CPU utilization of the Auto Scaling group to stay at around 50 percent when the load on the application changes. This gives you extra capacity to handle traffic spikes without maintaining an excessive number of idle resources.

You can meet this need by creating a target tracking scaling policy that targets an average CPU utilization of 50 percent. Then, your Auto Scaling group will scale out, or increase capacity, when CPU exceeds 50 percent to handle increased load. It will scale in, or decrease capacity, when CPU drops below 50 percent to optimize costs during periods of low utilization.

Topics

- Multiple target tracking scaling policies
- Choose metrics
- Define target value
- Define instance warmup time
- Considerations
- Create a target tracking scaling policy
- Create a target tracking scaling policy using metric math

Multiple target tracking scaling policies

To help optimize scaling performance, you can use multiple target tracking scaling policies together, provided that each of them uses a different metric. For example, utilization and throughput can influence each other. Whenever one of these metrics changes, it usually implies that other metrics will also be impacted. The use of multiple metrics therefore provides additional

information about the load that your Auto Scaling group is under. This can help Amazon EC2 Auto Scaling make more informed decisions when determining how much capacity to add to your group.

The intention of Amazon EC2 Auto Scaling is to always prioritize availability. It will scale out the Auto Scaling group if any of the target tracking policies are ready to scale out. It will scale in only if all of the target tracking policies (with the scale in portion enabled) are ready to scale in.

Choose metrics

You can create target tracking scaling policies with either predefined metrics or custom metrics.

When you create a target tracking scaling policy with a predefined metric type, you choose one metric from the following list of predefined metrics:

- `ASGAverageCPUUtilization`—Average CPU utilization of the Auto Scaling group.
- `ASGAverageNetworkIn`—Average number of bytes received by a single instance on all network interfaces.
- `ASGAverageNetworkOut`—Average number of bytes sent out from a single instance on all network interfaces.
- `ALBRequestCountPerTarget`—Average Application Load Balancer request count per target.

> **Important**
>
> Other valuable information about the metrics for CPU utilization, network I/O, and Application Load Balancer request count per target can be found in the List the available CloudWatch metrics for your instances topic in the *Amazon EC2 User Guide* and the CloudWatch metrics for your Application Load Balancer topic in the *User Guide for Application Load Balancers*, respectively.

You can choose other available CloudWatch metrics or your own metrics in CloudWatch by specifying a custom metric. You must use the AWS CLI or an SDK to create a target tracking policy with a customized metric specification. For an example that specifies a customized metric specification for a target tracking scaling policy using the AWS CLI, see Example scaling policies for the AWS CLI.

Keep the following in mind when choosing a metric:

- We recommend that you only use metrics that are available at one-minute intervals to help you scale faster in response to utilization changes. Target tracking will evaluate metrics aggregated at a one-minute granularity for all predefined metrics and custom metrics, but the underlying metric might publish data less frequently. For example, all Amazon EC2 metrics are sent in five-minute intervals by default, but they are configurable to one minute (known as detailed monitoring). This choice is up to the individual services. Most try to use the smallest interval possible. For information about enabling detailed monitoring, see Configure monitoring for Auto Scaling instances.
- Not all custom metrics work for target tracking. The metric must be a valid utilization metric and describe how busy an instance is. The metric value must increase or decrease proportionally to the number of instances in the Auto Scaling group. That's so the metric data can be used to proportionally scale out or in the number of instances. For example, the CPU utilization of an Auto Scaling group works (that is, the Amazon EC2 metric `CPUUtilization` with the metric dimension `AutoScalingGroupName`), if the load on the Auto Scaling group is distributed across the instances.
- The following metrics do not work for target tracking:
  - The number of requests received by the load balancer fronting the Auto Scaling group (that is, the Elastic Load Balancing metric `RequestCount`). The number of requests received by the load balancer doesn't change based on the utilization of the Auto Scaling group.
  - Load balancer request latency (that is, the Elastic Load Balancing metric `Latency`). Request latency can increase based on increasing utilization, but doesn't necessarily change proportionally.
  - The CloudWatch Amazon SQS queue metric `ApproximateNumberOfMessagesVisible`. The number of messages in a queue might not change proportionally to the size of the Auto Scaling group that processes messages from the queue. However, a custom metric that measures the number of messages in the queue per EC2 instance in the Auto Scaling group can work. For more information, see Scaling policy based on Amazon SQS.
- To use the `ALBRequestCountPerTarget` metric, you must specify the `ResourceLabel` parameter to identify the load balancer target group that is associated with the metric. For an example that specifies the `ResourceLabel` parameter for a target tracking scaling policy using the AWS CLI, see Example scaling policies for the AWS CLI.
- When a metric emits real 0 values to CloudWatch (for example, `ALBRequestCountPerTarget`), an Auto Scaling group can scale in to 0 when there is no traffic to your application for a sustained period of time. To have your Auto Scaling group scale in to 0 when no requests are routed it, the group's minimum capacity must be set to 0.

- Instead of publishing new metrics to use in your scaling policy, you can use metric math to combine existing metrics. For more information, see Create a target tracking scaling policy using metric math.

Define target value

When you create a target tracking scaling policy, you must specify a target value. The target value represents the optimal average utilization or throughput for the Auto Scaling group. To use resources cost efficiently, set the target value as high as possible with a reasonable buffer for unexpected traffic increases. When your application is optimally scaled out for a normal traffic flow, the actual metric value should be at or just below the target value.

When a scaling policy is based on throughput, such as the request count per target for an Application Load Balancer, network I/O, or other count metrics, the target value represents the optimal average throughput from a single instance, for a one-minute period.

Define instance warmup time

You can optionally specify the number of seconds that it takes for a newly launched instance to warm up. Until its specified warmup time has expired, an instance is not counted toward the aggregated EC2 instance metrics of the Auto Scaling group.

While instances are in the warmup period, your scaling policies only scale out if the metric value from instances that are not warming up is greater than the policy's target utilization.

If the group scales out again, the instances that are still warming up are counted as part of the desired capacity for the next scale-out activity. The intention is to continuously (but not excessively) scale out.

While the scale-out activity is in progress, all scale in activities initiated by scaling policies are blocked until the instances finish warming up. When the instances finish warming up, if a scale in event occurs, any instances currently in the process of terminating will be counted towards the current capacity of the group when calculating the new desired capacity. Therefore, we don't remove more instances from the Auto Scaling group than necessary.

Default value

If no value is set, then the scaling policy will use the default value, which is the value for the default instance warmup defined for the group. If the default instance warmup is null, then it falls

back to the value of the default cooldown. We recommend using the default instance warmup to make it easier to update all scaling policies when the warmup time changes.

Considerations

The following considerations apply when working with target tracking scaling policies:

- Do not create, edit, or delete the CloudWatch alarms that are used with a target tracking scaling policy. Amazon EC2 Auto Scaling creates and manages the CloudWatch alarms that are associated with your target tracking scaling policies and deletes them when no longer needed.
- A target tracking scaling policy prioritizes availability during periods of fluctuating traffic levels by scaling in more gradually when traffic is decreasing. If you want your Auto Scaling group to scale in immediately when a workload finishes, you can disable the scale-in portion of the policy. This provides you with the flexibility to use the scale-in method that best meets your needs when utilization is low. To ensure that scale in happens as quickly as possible, we recommend not using a simple scaling policy to prevent a cooldown period from being added.
- If the metric is missing data points, this causes the CloudWatch alarm state to change to `INSUFFICIENT_DATA`. When this happens, Amazon EC2 Auto Scaling cannot scale your group until new data points are found.
- If the metric is sparsely reported by design, metric math can be helpful. For example, to use the most recent values, then use the `FILL(m1,REPEAT)` function where `m1` is the metric.
- You might see gaps between the target value and the actual metric data points. This is because we act conservatively by rounding up or down when determining how many instances to add or remove. This prevents us from adding an insufficient number of instances or removing too many instances. However, for smaller Auto Scaling groups with fewer instances, the utilization of the group might seem far from the target value. For example, let's say that you set a target value of 50 percent for CPU utilization and your Auto Scaling group then exceeds the target. We might determine that adding 1.5 instances will decrease the CPU utilization to close to 50 percent. Because it is not possible to add 1.5 instances, we round up and add two instances. This might decrease the CPU utilization to a value below 50 percent, but it ensures that your application has enough resources to support it. Similarly, if we determine that removing 1.5 instances increases your CPU utilization to above 50 percent, we remove just one instance.

  For larger Auto Scaling groups with more instances, the utilization is spread over a larger number of instances, in which case adding or removing instances causes less of a gap between the target value and the actual metric data points.

- A target tracking scaling policy assumes that it should scale out your Auto Scaling group when the specified metric is above the target value. You can't use a target tracking scaling policy to scale out your Auto Scaling group when the specified metric is below the target value.

Create a target tracking scaling policy

To create a target tracking scaling policy for your Auto Scaling group, use one of the following methods.

Before you begin, confirm that your preferred metric is available at 1-minute intervals (compared to the default 5-minute interval of Amazon EC2 metrics).

Console

To create a target tracking scaling policy for a new Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Choose **Create Auto Scaling group**.
3. In Steps 1, 2, and 3, choose the options as desired and proceed to **Step 4: Configure group size and scaling policies**.
4. Under **Scaling**, specify the range that you want to scale between by updating the **Min desired capacity** and **Max desired capacity**. These two settings allow your Auto Scaling group to scale dynamically. For more information, see Set scaling limits for your Auto Scaling group.
5. Under **Automatic scaling**, choose **Target tracking scaling policy**.
6. To define a policy, do the following:

   a. Specify a name for the policy.

   b. For **Metric type**, choose a metric.

      If you chose **Application Load Balancer request count per target**, choose a target group in **Target group**.

   c. Specify a **Target value** for the metric.

   d. (Optional) For **Instance warmup**, update the instance warmup value as needed.

   e. (Optional) Select **Disable scale in to create only a scale-out policy**. This allows you to create a separate scale-in policy of a different type if wanted.

7. Proceed to create the Auto Scaling group. Your scaling policy will be created after the Auto Scaling group has been created.

To create a target tracking scaling policy for an existing Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. Verify that the scaling limits are appropriately set. For example, if your group's desired capacity is already at its maximum, you need to specify a new maximum in order to scale out. For more information, see Set scaling limits for your Auto Scaling group.
4. On the **Automatic scaling** tab, in **Dynamic scaling policies**, choose **Create dynamic scaling policy**.
5. To define a policy, do the following:

   a. For **Policy type**, keep the default of **Target tracking scaling**.

   b. Specify a name for the policy.

   c. For **Metric type**, choose a metric. You can choose only one metric type. To use more than one metric, create multiple policies.

      If you chose **Application Load Balancer request count per target**, choose a target group in **Target group**.

   d. Specify a **Target value** for the metric.

   e. (Optional) For **Instance warmup**, update the instance warmup value as needed.

   f. (Optional) Select **Disable scale in to create only a scale-out policy**. This allows you to create a separate scale-in policy of a different type if wanted.
6. Choose **Create**.

AWS CLI

To create a target tracking scaling policy, you can use the following example to help you get started. Replace each *`user input placeholder`* with your own information.

> **Note**
>
> For more examples, see Example scaling policies for the AWS CLI.

To create a target tracking scaling policy (AWS CLI)

1. Use the following `cat` command to store a target value for your scaling policy and a predefined metric specification in a JSON file named `config.json` in your home directory. The following is an example target tracking configuration that keeps the average CPU utilization at 50 percent.

```
$ cat ~/config.json
{
  "TargetValue": 50.0,
  "PredefinedMetricSpecification":
    {
      "PredefinedMetricType": "ASGAverageCPUUtilization"
    }
}
```

   For more information, see PredefinedMetricSpecification in the *Amazon EC2 Auto Scaling API Reference*.

2. Use the put-scaling-policy command, along with the `config.json` file that you created in the previous step, to create your scaling policy.

```
aws autoscaling put-scaling-policy --policy-name cpu50-target-tracking-scaling-
policy \
  --auto-scaling-group-name my-asg --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://config.json
```

   If successful, this command returns the ARNs and names of the two CloudWatch alarms created on your behalf.

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
west-2:123456789012:scalingPolicy:228f02c2-c665-4bfd-
aaac-8b04080bea3c:autoScalingGroupName/my-asg:policyName/cpu50-target-tracking-
scaling-policy",
```

```
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e",
            "AlarmName": "TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e"
        },
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2",
            "AlarmName": "TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2"
        }
    ]
}
```

Create a target tracking scaling policy using metric math

Using metric math, you can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series in the CloudWatch console and add them to dashboards. For more information about metric math, see Using metric math in the *Amazon CloudWatch User Guide*.

The following considerations apply to metric math expressions:

- You can query any available CloudWatch metric. Each metric is a unique combination of metric name, namespace, and zero or more dimensions.
- You can use any arithmetic operator (+ - * / ^), statistical function (such as AVG or SUM), or other function that CloudWatch supports.
- You can use both metrics and the results of other math expressions in the formulas of the math expression.
- Any expressions used in a metric specification must eventually return a single time series.
- You can verify that a metric math expression is valid by using the CloudWatch console or the CloudWatch GetMetricData API.

> **Note**
>
> You can create a target tracking scaling policy using metric math only if you use the AWS CLI, AWS CloudFormation, or an SDK. This feature is not yet available in the console.

Example: Amazon SQS queue backlog per instance

To calculate the Amazon SQS queue backlog per instance, take the approximate number of messages available for retrieval from the queue and divide that number by the Auto Scaling group's running capacity, which is the number of instances in the `InService` state. For more information, see Scaling policy based on Amazon SQS.

The logic for the expression is this:

```
sum of (number of messages in the queue)/(number of InService instances)
```

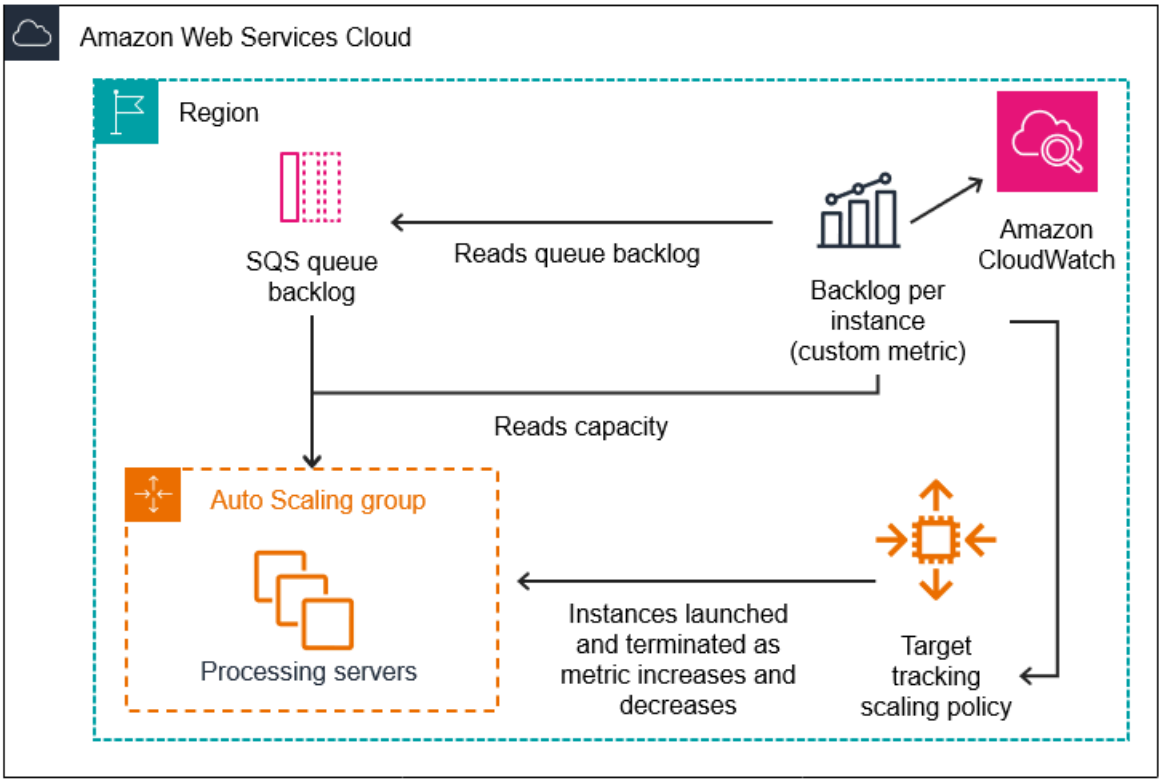
Then your CloudWatch metric information is the following.

ID	CloudWatch metric	Statistic	Period
m1	ApproximateNumberOfMessagesVisible	Sum	1 minute
m2	GroupInServiceInstances	Average	1 minute

Your metric math ID and expression are the following.

ID	Expression
e1	`(m1)/(m2)`

The following diagram illustrates the architecture for this metric:



To use this metric math to create a target tracking scaling policy (AWS CLI)

1. Store the metric math expression as part of a customized metric specification in a JSON file named `config.json`.

   Use the following example to help you get started. Replace each *`user input placeholder`* with your own information.

```
{
    "CustomizedMetricSpecification": {
        "Metrics": [
            {
                "Label": "Get the queue size (the number of messages waiting to be
 processed)",
                "Id": "m1",
                "MetricStat": {
                    "Metric": {
                        "MetricName": "ApproximateNumberOfMessagesVisible",
                        "Namespace": "AWS/SQS",
                        "Dimensions": [
                            {
                                "Name": "QueueName",
```

```
                                "Value": "my-queue"
                            }
                        ]
                    },
                    "Stat": "Sum"
                },
                "ReturnData": false
            },
            {
                "Label": "Get the group size (the number of InService instances)",
                "Id": "m2",
                "MetricStat": {
                    "Metric": {
                        "MetricName": "GroupInServiceInstances",
                        "Namespace": "AWS/AutoScaling",
                        "Dimensions": [
                            {
                                "Name": "AutoScalingGroupName",
                                "Value": "my-asg"
                            }
                        ]
                    },
                    "Stat": "Average"
                },
                "ReturnData": false
            },
            {
                "Label": "Calculate the backlog per instance",
                "Id": "e1",
                "Expression": "m1 / m2",
                "ReturnData": true
            }
        ]
    },
    "TargetValue": 100
}
```

For more information, see TargetTrackingConfiguration in the *Amazon EC2 Auto Scaling API Reference*.

> **Note**
>
> Following are some additional resources that can help you find metric names, namespaces, dimensions, and statistics for CloudWatch metrics:
>
> - For information about the available metrics for AWS services, see AWS services that publish CloudWatch metrics in the *Amazon CloudWatch User Guide*.
> - To get the exact metric name, namespace, and dimensions (if applicable) for a CloudWatch metric with the AWS CLI, see list-metrics.

2. To create this policy, run the put-scaling-policy command using the JSON file as input, as demonstrated in the following example.

```
aws autoscaling put-scaling-policy --policy-name sqs-backlog-target-tracking-
scaling-policy \
  --auto-scaling-group-name my-asg --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN) and the ARNs of the two CloudWatch alarms created on your behalf.

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
west-2:123456789012:scalingPolicy:228f02c2-c665-4bfd-
aaac-8b04080bea3c:autoScalingGroupName/my-asg:policyName/sqs-backlog-target-
tracking-scaling-policy",
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e",
            "AlarmName": "TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e"
        },
        {
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2",
            "AlarmName": "TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2"
```

```
        }
    ]
}
```

> **Note**
>
> If this command throws an error, make sure that you have updated the AWS CLI locally to the latest version.

Step and simple scaling policies for Amazon EC2 Auto Scaling

Step scaling and simple scaling policies scale the capacity of your Auto Scaling group in predefined increments based on CloudWatch alarms. You can define separate scaling policies to handle scaling out (increasing capacity) and scaling in (decreasing capacity) when an alarm threshold is breached.

With step scaling and simple scaling, you create and manage the CloudWatch alarms that invoke the scaling process. When an alarm is breached, Amazon EC2 Auto Scaling initiates the scaling policy associated with that alarm.

We strongly recommend that you use target tracking scaling policies to scale on metrics like average CPU utilization or average request count per target. Metrics that decrease when capacity increases and increase when capacity decreases can be used to proportionally scale out or in the number of instances using target tracking. This helps ensure that Amazon EC2 Auto Scaling follows the demand curve for your applications closely. For more information, see Target tracking scaling policies.

Contents

- How step scaling policies work
- Step adjustments for step scaling
- Scaling adjustment types
- Instance warmup
- Considerations
- Create a step scaling policy for scale out
- Create a step scaling policy for scale in
- Simple scaling policies

How step scaling policies work

To use step scaling, you first create a CloudWatch alarm that monitors a metric for your Auto Scaling group. Define the metric, threshold value, and number of evaluation periods that determine an alarm breach. Then, create a step scaling policy that defines how to scale your group when the alarm threshold is breached.

Add the step adjustments in the policy. You can define different step adjustments based on the breach size of the alarm. For example:

- Scale out by 10 instances if the alarm metric reaches 60 percent
- Scale out by 30 instances if the alarm metric reaches 75 percent
- Scale out by 40 instances if the alarm metric reaches 85 percent

When the alarm threshold is breached for the specified number of evaluation periods, Amazon EC2 Auto Scaling will apply the step adjustments defined in the policy. The adjustments can continue for additional alarm breaches until the alarm state returns to OK.

Each instance has a warmup period to prevent scaling activities from being too reactive to changes that occur over short periods of time. You can optionally configure the warmup period for your scaling policy. However, we recommend using the default instance warmup to make it easier to update all scaling policies when the warmup time changes. For more information, see Set the default instance warmup for an Auto Scaling group.

Simple scaling policies are similar to step scaling policies, except they're based on a single scaling adjustment, with a cooldown period between each scaling activity. For more information, see Simple scaling policies.

Step adjustments for step scaling

When you create a step scaling policy, you specify one or more step adjustments that automatically scale the number of instances dynamically based on the size of the alarm breach. Each step adjustment specifies the following:

- A lower bound for the metric value
- An upper bound for the metric value
- The amount by which to scale, based on the scaling adjustment type

CloudWatch aggregates metric data points based on the statistic for the metric that's associated with your CloudWatch alarm. When the alarm is breached, the appropriate scaling policy is invoked. Amazon EC2 Auto Scaling applies the aggregation type to the most recent metric data points from CloudWatch (as opposed to the raw metric data). It compares this aggregated metric value against the upper and lower bounds defined by the step adjustments to determine which step adjustment to perform.

You specify the upper and lower bounds relative to the breach threshold. For example, let's say you made a CloudWatch alarm and a scale-out policy for when the metric is above 50 percent. You then made a second alarm and a scale-in policy for when the metric is below 50 percent. You made a set of step adjustments with an adjustment type of `PercentChangeInCapacity` (or **Percent of group** in the console) for each policy:

Example: Step adjustments for scale-out policy

Lower bound	Upper bound	Adjustment
0	10	0
10	20	10
20	null	30

Example: Step adjustments for scale-in policy

Lower bound	Upper bound	Adjustment
-10	0	0
-20	-10	-10
null	-20	-30

This creates the following scaling configuration.

```
Metric value

-infinity          30%    40%          60%     70%             infinity
-----------------------------------------------------------------------
          -30%      | -10% | Unchanged  | +10%  |       +30%
```

```
-----------------------------------------------------------------------
```

Now, let's say that you use this scaling configuration on an Auto Scaling group that has both a current capacity and a desired capacity of 10. The following points summarize the behavior of the scaling configuration in relation to the desired and current capacity of the group:

- The desired and current capacity is maintained while the aggregated metric value is greater than 40 and less than 60.
- If the metric value gets to 60, the desired capacity of the group increases by 1 instance, to 11 instances, based on the second step adjustment of the scale-out policy (add 10 percent of 10 instances). After the new instance is running and its specified warmup time has expired, the current capacity of the group increases to 11 instances. If the metric value rises to 70 even after this increase in capacity, the desired capacity of the group increases by another 3 instances, to 14 instances. This is based on the third step adjustment of the scale-out policy (add 30 percent of 11 instances, 3.3 instances, rounded down to 3 instances).
- If the metric value gets to 40, the desired capacity of the group decreases by 1 instance, to 13 instances, based on the second step adjustment of the scale-in policy (remove 10 percent of 14 instances, 1.4 instances, rounded down to 1 instance). If the metric value falls to 30 even after this decrease in capacity, the desired capacity of the group decreases by another 3 instances, to 10 instances. This is based on the third step adjustment of the scale-in policy (remove 30 percent of 13 instances, 3.9 instances, rounded down to 3 instances).

When you specify the step adjustments for your scaling policy, note the following:

- If you use the AWS Management Console, you specify the upper and lower bounds as absolute values. If you use the AWS CLI or an SDK, you specify the upper and lower bounds relative to the breach threshold.
- The ranges of your step adjustments can't overlap or have a gap.
- Only one step adjustment can have a null lower bound (negative infinity). If one step adjustment has a negative lower bound, then there must be a step adjustment with a null lower bound.
- Only one step adjustment can have a null upper bound (positive infinity). If one step adjustment has a positive upper bound, then there must be a step adjustment with a null upper bound.
- The upper and lower bound can't be null in the same step adjustment.
- If the metric value is above the breach threshold, the lower bound is inclusive and the upper bound is exclusive. If the metric value is below the breach threshold, the lower bound is exclusive and the upper bound is inclusive.

Scaling adjustment types

You can define a scaling policy that performs the optimal scaling action, based on the scaling adjustment type that you choose. You can specify the adjustment type as a percentage of the current capacity of your Auto Scaling group, or in capacity units. Normally a capacity unit means one instance, unless you are using the instance weights feature.

Amazon EC2 Auto Scaling supports the following adjustment types for step scaling and simple scaling:

- `ChangeInCapacity` — Increment or decrement the current capacity of the group by the specified value. A positive value increases the capacity and a negative adjustment value decreases the capacity. For example: If the current capacity of the group is 3 and the adjustment is 5, then when this policy is performed, we add 5 capacity units to the capacity for a total of 8 capacity units.
- `ExactCapacity` — Change the current capacity of the group to the specified value. Specify a non-negative value with this adjustment type. For example: If the current capacity of the group is 3 and the adjustment is 5, then when this policy is performed, we change the capacity to 5 capacity units.
- `PercentChangeInCapacity` — Increment or decrement the current capacity of the group by the specified percentage. A positive value increases the capacity and a negative value decreases the capacity. For example: If the current capacity is 10 and the adjustment is 10 percent, then when this policy is performed, we add 1 capacity unit to the capacity for a total of 11 capacity units.

> **Note**
>
> If the resulting value is not an integer, it is rounded as follows:
>
> - Values greater than 1 are rounded down. For example, `12.7` is rounded to 12.
> - Values between 0 and 1 are rounded to 1. For example, `.67` is rounded to 1.
> - Values between 0 and -1 are rounded to -1. For example, `-.58` is rounded to `-1`.
> - Values less than -1 are rounded up. For example, `-6.67` is rounded to `-6`.

With `PercentChangeInCapacity`, you can also specify the minimum number of instances to scale using the `MinAdjustmentMagnitude` parameter. For example, suppose that you create a policy that adds 25 percent and you specify a minimum increment of 2 instances. If you have

an Auto Scaling group with 4 instances and the scaling policy is executed, 25 percent of 4 is 1 instance. However, because you specified a minimum increment of 2, there are 2 instances added.

When you use instance weights, the effect of setting the `MinAdjustmentMagnitude` parameter to a non-zero value changes. The value is in capacity units. To set the minimum number of instances to scale, set this parameter to a value that is at least as large as your largest instance weight.

If you use instance weights, keep in mind that the current capacity of your Auto Scaling group can exceed the desired capacity as needed. If your absolute number to decrement, or the amount that the percentage says to decrement, is less than the difference between current and desired capacity, no scaling action is taken. You must take these behaviors into account when you look at the outcome of a scaling policy when a threshold alarm is in breach. For example, suppose that the desired capacity is 30 and the current capacity is 32. When the alarm is in breach, if the scaling policy decrements the desired capacity by 1, then no scaling action is taken.

Instance warmup

For step scaling, you can optionally specify the number of seconds that it takes for a newly launched instance to warm up. Until its specified warmup time has expired, an instance is not counted toward the aggregated EC2 instance metrics of the Auto Scaling group.

While instances are in the warmup period, your scaling policies only scale out if the metric value from instances that are not warming up is greater than the policy's alarm high threshold.

If the group scales out again, the instances that are still warming up are counted as part of the desired capacity for the next scale-out activity. Therefore, multiple alarm breaches that fall in the range of the same step adjustment result in a single scaling activity. The intention is to continuously (but not excessively) scale out.

For example, let's say that you create a policy with two steps. The first step adds 10 percent when the metric gets to 60, and the second step adds 30 percent when the metric gets to 70 percent. Your Auto Scaling group has a desired and current capacity of 10. The desired and current capacity do not change while the aggregated metric value is less than 60. Suppose that the metric gets to 60, so 1 instance is added (10 percent of 10 instances). Then, the metric gets to 62 while the new instance is still warming up. The scaling policy calculates the new desired capacity based on the current capacity, which is still 10. However, the desired capacity of the group has already increased to 11 instances, so the scaling policy does not increase the desired capacity further. If the metric gets to 70 while the new instance is still warming up, we should add 3 instances (30 percent of 10

instances). However, the desired capacity of the group is already 11, so we add only 2 instances, for a new desired capacity of 13 instances.

While the scale-out activity is in progress, all scale-in activities initiated by scaling policies are blocked until the instances finish warming up. When the instances finish warming up, if a scale-in event occurs, any instances currently in the process of terminating will be counted towards the current capacity of the group when calculating the new desired capacity. Therefore, we don't remove more instances from the Auto Scaling group than necessary. For example, while an instance is already terminating, if an alarm is in breach in the range of the same step adjustment that decremented the desired capacity by 1, then no scaling action is taken.

Default value

If no value is set, then the scaling policy will use the default value, which is the value for the default instance warmup defined for the group. If the default instance warmup is null, then it falls back to the value of the default cooldown.

Considerations

The following considerations apply when working with step and simple scaling policies:

- Consider whether you can predict the step adjustments on the application accurately enough to use step scaling. If your scaling metric increases or decreases proportionally to the capacity of the scalable target, we recommend that you use a target tracking scaling policy instead. You still have the option to use step scaling as an additional policy for a more advanced configuration. For example, you can configure a more aggressive response when utilization reaches a certain level.
- Make sure to choose an adequate margin between the scale-out and scale-in thresholds to prevent flapping. Flapping is an infinite loop of scaling in and scaling out. That is, if a scaling action is taken, the metric value would change and start another scaling action in the reverse direction.

Create a step scaling policy for scale out

To create a step scaling policy for scale out for your Auto Scaling group, use one of the following methods:

Console

Step 1: Create a CloudWatch alarm for the metric high threshold

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the Region. From the navigation bar, select the Region where your Auto Scaling group resides.
3. In the navigation pane, choose **Alarms, All alarms** and then choose **Create alarm**.
4. Choose **Select metric**.
5. On the **All metrics** tab, choose **EC2**, **By Auto Scaling Group**, and enter the Auto Scaling group's name in the search field. Then, select `CPUUtilization` and choose **Select metric**. The **Specify metric and conditions** page appears, showing a graph and other information about the metric.
6. For **Period**, choose the evaluation period for the alarm, for example, 1 minute. When evaluating the alarm, each period is aggregated into one data point.

   > **Note**
   >
   > A shorter period creates a more sensitive alarm.

7. Under **Conditions**, do the following:

   - For **Threshold type**, choose **Static**.
   - For **Whenever `CPUUtilization` is**, specify whether you want the value of the metric to be greater than or greater than or equal to the threshold to breach the alarm. Then, under **than**, enter the threshold value that you want to breach the alarm.

   > **Important**
   >
   > For an alarm to use with a scale-out policy (metric high), make sure you do not choose less than or less than or equal to the threshold.

8. Under **Additional configuration**, do the following:

   - For **Datapoints to alarm**, enter the number of data points (evaluation periods) during which the metric value must meet the threshold conditions for the alarm. For example, two consecutive periods of 5 minutes would take 10 minutes to invoke the alarm state.

- For **Missing data treatment**, choose **Treat missing data as bad (breaching threshold)**. For more information, see Configuring how CloudWatch alarms treat missing data in the *Amazon CloudWatch User Guide*.

9. Choose **Next**.

   The **Configure actions** page appears.

10. Under **Notification**, select an Amazon SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

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

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

11. You can leave the other sections of the **Configure actions** page empty. Leaving the other sections empty creates an alarm without associating it to a scaling policy. You can then associate the alarm with a scaling policy from the Amazon EC2 Auto Scaling console.
12. Choose **Next**.
13. Enter a name (for example, `Step-Scaling-AlarmHigh-AddCapacity`) and, optionally, a description for the alarm, and then choose **Next**.
14. Choose **Create alarm**.

Use the following procedure to continue where you left off after creating your CloudWatch alarm.

Step 2: Create a step scaling policy for scale out

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.

3. Verify that the scaling limits are appropriately set. For example, if your group's desired capacity is already at its maximum, you need to specify a new maximum in order to scale out. For more information, see Set scaling limits for your Auto Scaling group.
4. On the **Automatic scaling** tab, in **Dynamic scaling policies**, choose **Create dynamic scaling policy**.

5. For **Policy type**, choose **Step scaling**, and then specify a name for the policy.
6. For **CloudWatch alarm**, choose your alarm. If you haven't already created an alarm, choose **Create a CloudWatch alarm** and complete step 4 through step 14 in the previous procedure to create an alarm.
7. Specify the change in the current group size that this policy will make when executed using **Take the action**. You can add a specific number of instances or a percentage of the existing group size, or set the group to an exact size.

   For example, to create a scale-out policy that increases the capacity of the group by 30 percent, choose Add, enter `30` in the next field, and then choose `percent of group`. By default, the lower bound for this step adjustment is the alarm threshold and the upper bound is positive (+) infinity.
8. To add another step, choose **Add step** and then define the amount by which to scale and the lower and upper bounds of the step relative to the alarm threshold.
9. To set a minimum number of instances to scale, update the number field in **Add capacity units in increments of at least** `1` **capacity units**.
10. (Optional) For **Instance warmup**, update the instance warmup value as needed.
11. Choose **Create**.

AWS CLI

To create a step scaling policy for scale out (increase capacity), you can use the following example commands. Replace each *`user input placeholder`* with your own information.

When you use the AWS CLI, you first create a step scaling policy that provides instructions to Amazon EC2 Auto Scaling about how to scale out when a metric's value is increasing. Then, you create the alarm by identifying the metric to watch, defining the metric high threshold and other details for the alarms, and associating the alarm with the scaling policy.

Step 1: Create a policy for scale out

Use the following put-scaling-policy command to create a step scaling policy named `my-step-scale-out-policy`, with an adjustment type of `PercentChangeInCapacity` that increases the capacity of the group based on the following step adjustments (assuming a CloudWatch alarm threshold of 60 percent):

- Increase the instance count by 10 percent when the value of the metric is greater than or equal to 60 percent but less than 75 percent
- Increase the instance count by 20 percent when the value of the metric is greater than or equal to 75 percent but less than 85 percent
- Increase the instance count by 30 percent when the value of the metric is greater than or equal to 85 percent

```
aws autoscaling put-scaling-policy \
  --auto-scaling-group-name my-asg  \
  --policy-name my-step-scale-out-policy \
  --policy-type StepScaling \
  --adjustment-type PercentChangeInCapacity \
  --metric-aggregation-type Average \
  --step-adjustments
 MetricIntervalLowerBound=0.0,MetricIntervalUpperBound=15.0,ScalingAdjustment=10 \

 MetricIntervalLowerBound=15.0,MetricIntervalUpperBound=25.0,ScalingAdjustment=20 \
                     MetricIntervalLowerBound=25.0,ScalingAdjustment=30 \
  --min-adjustment-magnitude 1
```

Record the policy's Amazon Resource Name (ARN). You need it to create a CloudWatch alarm for the policy.

```
{
    "PolicyARN":
 "arn:aws:autoscaling:region:123456789012:scalingPolicy:4ee9e543-86b5-4121-b53b-
aa4c23b5bbcc:autoScalingGroupName/my-asg:policyName/my-step-scale-in-policy
}
```

Step 2: Create a CloudWatch alarm for the metric high threshold

Use the following CloudWatch put-metric-alarm command to create an alarm that increases the size of the Auto Scaling group based on an average CPU threshold value of 60 percent for at least two consecutive evaluation periods of two minutes. To use your own custom metric, specify its name in `--metric-name` and its namespace in `--namespace`.

```
aws cloudwatch put-metric-alarm --alarm-name Step-Scaling-AlarmHigh-AddCapacity \
  --metric-name CPUUtilization --namespace AWS/EC2 --statistic Average \
  --period 120 --evaluation-periods 2 --threshold 60 \
```

```
--comparison-operator GreaterThanOrEqualToThreshold \
--dimensions "Name=AutoScalingGroupName,Value=my-asg" \
--alarm-actions PolicyARN
```

Create a step scaling policy for scale in

To create a step scaling policy for scale in for your Auto Scaling group, use one of the following methods:

Console

Step 1: Create a CloudWatch alarm for the metric low threshold

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the Region. From the navigation bar, select the Region where your Auto Scaling group resides.
3. In the navigation pane, choose **Alarms, All alarms** and then choose **Create alarm**.
4. Choose **Select metric**.
5. On the **All metrics** tab, choose **EC2**, **By Auto Scaling Group**, and enter the Auto Scaling group's name in the search field. Then, select `CPUUtilization` and choose **Select metric**. The **Specify metric and conditions** page appears, showing a graph and other information about the metric.
6. For **Period**, choose the evaluation period for the alarm, for example, 1 minute. When evaluating the alarm, each period is aggregated into one data point.

   > **Note**
   >
   > A shorter period creates a more sensitive alarm.

7. Under **Conditions**, do the following:

   - For **Threshold type**, choose **Static**.
   - For **Whenever `CPUUtilization` is**, specify whether you want the value of the metric to be less than or less than or equal to the threshold to breach the alarm. Then, under **than**, enter the threshold value that you want to breach the alarm.

> **Important**
>
> For an alarm to use with a scale in policy (metric low), make sure you do not choose greater than or greater than or equal to the threshold.

8. Under **Additional configuration**, do the following:
   - For **Datapoints to alarm**, enter the number of data points (evaluation periods) during which the metric value must meet the threshold conditions for the alarm. For example, two consecutive periods of 5 minutes would take 10 minutes to invoke the alarm state.
   - For **Missing data treatment**, choose **Treat missing data as bad (breaching threshold)**. For more information, see Configuring how CloudWatch alarms treat missing data in the *Amazon CloudWatch User Guide*.
9. Choose **Next**.

   The **Configure actions** page appears.
10. Under **Notification**, select an Amazon SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

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

    To have the alarm not send notifications, choose **Remove**.
11. You can leave the other sections of the **Configure actions** page empty. Leaving the other sections empty creates an alarm without associating it to a scaling policy. You can then associate the alarm with a scaling policy from the Amazon EC2 Auto Scaling console.
12. Choose **Next**.
13. Enter a name (for example, `Step-Scaling-AlarmLow-RemoveCapacity`) and, optionally, a description for the alarm, and then choose **Next**.
14. Choose **Create alarm**.

Use the following procedure to continue where you left off after creating your CloudWatch alarm.

Step 2: Create a step scaling policy for scale in

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. Verify that the scaling limits are appropriately set. For example, if your group's desired capacity is already at its minimum, you need to specify a new minimum in order to scale in. For more information, see Set scaling limits for your Auto Scaling group.
4. On the **Automatic scaling** tab, in **Dynamic scaling policies**, choose **Create dynamic scaling policy**.
5. For **Policy type**, choose **Step scaling**, and then specify a name for the policy.
6. For **CloudWatch alarm**, choose your alarm. If you haven't already created an alarm, choose **Create a CloudWatch alarm** and complete step 4 through step 14 in the previous procedure to create an alarm.
7. Specify the change in the current group size that this policy will make when executed using **Take the action**. You can remove a specific number of instances or a percentage of the existing group size, or set the group to an exact size.

   For example, to create a scale in policy that decreases the capacity of the group by two instances, choose `Remove`, enter 2 in the next field, and then choose `capacity units`. By default, the upper bound for this step adjustment is the alarm threshold and the lower bound is negative (-) infinity.
8. To add another step, choose **Add step** and then define the amount by which to scale and the lower and upper bounds of the step relative to the alarm threshold.
9. Choose **Create**.

AWS CLI

To create a step scaling policy for scale in (decrease capacity), you can use the following example commands. Replace each *`user input placeholder`* with your own information.

When you use the AWS CLI, you first create a step scaling policy that provides instructions to Amazon EC2 Auto Scaling about how to scale in when a metric's value is decreasing. Then, you

create the alarm by identifying the metric to watch, defining the metric low threshold and other details for the alarms, and associating the alarm with the scaling policy.

Step 1: Create a policy for scale in

Use the following put-scaling-policy command to create a step scaling policy named `my-step-scale-in-policy`, with an adjustment type of `ChangeInCapacity` that decreases the capacity of the group by 2 instances when the associated CloudWatch alarm breaches the metric low threshold value.

```
aws autoscaling put-scaling-policy \
  --auto-scaling-group-name my-asg  \
  --policy-name my-step-scale-in-policy \
  --policy-type StepScaling \
  --adjustment-type ChangeInCapacity \
  --step-adjustments MetricIntervalUpperBound=0.0,ScalingAdjustment=-2
```

Record the policy's Amazon Resource Name (ARN). You need it to create the CloudWatch alarm for the policy.

```
{
    "PolicyARN": "arn:aws:autoscaling:region:123456789012:scalingPolicy:ac542982-cbeb-4294-891c-a5a941dfa787:autoScalingGroupName/my-asg:policyName/my-step-scale-out-policy
}
```

Step 2: Create a CloudWatch alarm for the metric low threshold

Use the following CloudWatch put-metric-alarm command to create an alarm that decreases the size of the Auto Scaling group based on average CPU threshold value of 40 percent for at least two consecutive evaluation periods of two minutes. To use your own custom metric, specify its name in `--metric-name` and its namespace in `--namespace`.

```
aws cloudwatch put-metric-alarm --alarm-name Step-Scaling-AlarmLow-RemoveCapacity \
  --metric-name CPUUtilization --namespace AWS/EC2 --statistic Average \
  --period 120 --evaluation-periods 2 --threshold 40 \
  --comparison-operator LessThanOrEqualToThreshold \
  --dimensions "Name=AutoScalingGroupName,Value=my-asg" \
  --alarm-actions PolicyARN
```

Simple scaling policies

The following examples show how you can use CLI commands to create simple scaling policies. They remain in this document as a reference for any customers who want to use them, but we recommend that you use target tracking or step scaling policies instead.

Similar to step scaling policies, simple scaling policies require you to create CloudWatch alarms for your scaling policies. In the policies that you create, you must also define whether to add or remove instances, and how many, or set the group to an exact size.

One of the main differences between step scaling policies and simple scaling policies is the step adjustments that you get with step scaling policies. With step scaling, you can make bigger or smaller changes to the size of the group based on the step adjustments that you specify.

A simple scaling policy must also wait for an in-progress scaling activity or health check replacement to complete and a cooldown period to end before it responds to additional alarms. In contrast, with step scaling, the policy continues to respond to additional alarms, even while a scaling activity or health check replacement is in progress. This means that Amazon EC2 Auto Scaling evaluates all alarm breaches as it receives the alarm messages. Because of this, we recommend that you use step scaling policies instead, even if you have only a single scaling adjustment.

Amazon EC2 Auto Scaling originally supported only simple scaling policies. If you created your scaling policy before target tracking and step scaling policies were introduced, your policy is treated as a simple scaling policy.

Create a simple scaling policy for scale out

Use the following put-scaling-policy command to create a simple scaling policy named `my-simple-scale-out-policy`, with an adjustment type of `PercentChangeInCapacity` that increases the capacity of the group by 30 percent when the associated CloudWatch alarm breaches the metric high threshold value.

```
aws autoscaling put-scaling-policy --policy-name my-simple-scale-out-policy \
  --auto-scaling-group-name my-asg --scaling-adjustment 30 \
  --adjustment-type PercentChangeInCapacity
```

Record the policy's Amazon Resource Name (ARN). You need it to create the CloudWatch alarm for the policy.

Create a simple scaling policy for scale in

Use the following put-scaling-policy command to create a simple scaling policy named `my-simple-scale-in-policy`, with an adjustment type of `ChangeInCapacity` that decreases the capacity of the group by one instance when the associated CloudWatch alarm breaches the metric low threshold value.

```
aws autoscaling put-scaling-policy --policy-name my-simple-scale-in-policy \
  --auto-scaling-group-name my-asg --scaling-adjustment -1 \
  --adjustment-type ChangeInCapacity --cooldown 180
```

Record the policy's Amazon Resource Name (ARN). You need it to create the CloudWatch alarm for the policy.

Scaling cooldowns for Amazon EC2 Auto Scaling

> **Important**
>
> As a best practice, we recommend that you do not use simple scaling policies and scaling cooldowns. A target tracking scaling policy or a step scaling policy is better for scaling performance. For a scaling policy that changes the size of your Auto Scaling group proportionally as the value of the scaling metric decreases or increases, we recommend target tracking over either simple scaling or step scaling.

When you create simple scaling policies for your Auto Scaling group, we recommend that you configure the scaling cooldown at the same time.

After your Auto Scaling group launches or terminates instances, it waits for a cooldown period to end before any further scaling activities initiated by simple scaling policies can start. The intention of the cooldown period is to let your Auto Scaling group stabilize and prevent it from launching or terminating additional instances before the effects of the previous scaling activity are visible.

Suppose, for example, that a simple scaling policy for CPU utilization recommends launching two instances. Amazon EC2 Auto Scaling launches two instances and then pauses the scaling activities until the cooldown period ends. After the cooldown period ends, any scaling activities initiated by simple scaling policies can resume. If CPU utilization breaches the alarm high threshold again, the Auto Scaling group scales out again, and the cooldown period takes effect again. However, if two instances were enough to bring the metric value back down, the group remains at its current size.

Contents

- Considerations
- Lifecycle hooks can cause additional delays
- Change the default cooldown period
- Set a cooldown period for specific simple scaling policies

Considerations

The following considerations apply when working with simple scaling policies and scaling cooldowns:

- Target tracking and step scaling policies can initiate a scale-out activity immediately without waiting for the cooldown period to end. Instead, whenever your Auto Scaling group launches instances, the individual instances have a warmup period. For more information, see Set the default instance warmup for an Auto Scaling group.
- When a scheduled action starts at the scheduled time, it can also initiate a scaling activity immediately without waiting for the cooldown period to end.
- If an instance becomes unhealthy, Amazon EC2 Auto Scaling does not wait for the cooldown period to end before replacing the unhealthy instance.
- When multiple instances launch or terminate, the cooldown period (either the default cooldown or the scaling policy-specific cooldown) takes effect starting when the last instance finishes launching or terminating.
- When you manually scale your Auto Scaling group, the default is not to wait for a cooldown to end. However, you can override this behavior and honor the default cooldown when you use the AWS CLI or an SDK to manually scale.
- By default, Elastic Load Balancing waits 300 seconds to complete the deregistration (connection draining) process. If the group is behind an Elastic Load Balancing load balancer, it will wait for the terminating instances to deregister before starting the cooldown period.

Lifecycle hooks can cause additional delays

If a lifecycle hook is invoked, the cooldown period begins after you complete the lifecycle action or after the timeout period ends. For example, consider an Auto Scaling group that has a lifecycle hook for instance launch. When the application experiences an increase in demand, the group launches an instance to add capacity. Because there is a lifecycle hook, the instance is put into a

wait state and scaling activities due to simple scaling policies are paused. When the instance enters the `InService` state, the cooldown period starts. When the cooldown period ends, simple scaling policy activities are resumed.

When Elastic Load Balancing is enabled, for the purposes of scaling in, the cooldown period starts when the instance that's selected for termination starts connection draining (deregistration delay). The cooldown period doesn't wait for connection draining to finish or the lifecycle hook to complete its action. This means that any scaling activities due to simple scaling policies can resume as soon as the result of the scale in event is reflected in the capacity of the group. Otherwise, waiting to complete all three activities—connection draining, a lifecycle hook, and a cooldown period— would significantly increase the amount of time that the Auto Scaling group needs to pause scaling.

Change the default cooldown period

You can't set the default cooldown when you initially create an Auto Scaling group in the Amazon EC2 Auto Scaling console. By default, this cooldown period is set to 300 seconds (5 minutes). If needed, you can update this after the group is created.

To change the default cooldown period (console)

After creating the Auto Scaling group, on the **Details** tab, choose **Advanced configurations**, **Edit**. For **Default cooldown**, choose the amount of time that you want based on your instance startup time or other application needs.

To change the default cooldown period (AWS CLI)

Use the following commands to change the default cooldown for new or existing Auto Scaling groups. If the default cooldown is not defined, the default value of 300 seconds is used.

- create-auto-scaling-group
- update-auto-scaling-group

To confirm the value of the default cooldown, use the describe-auto-scaling-groups command.

Set a cooldown period for specific simple scaling policies

By default, all simple scaling policies use the default cooldown period that is defined for the Auto Scaling group. To set a cooldown period for specific simple scaling policies, use the optional

cooldown parameter when you create or update the policy. When a cooldown period is specified for a policy, it overrides the default cooldown.

One common use for a scaling policy-specific cooldown period is with a scale in policy. Because this policy terminates instances, Amazon EC2 Auto Scaling needs less time to determine whether to terminate additional instances. Terminating instances should be a much quicker operation than launching instances. The default cooldown period of 300 seconds is therefore too long. In this case, a scaling policy-specific cooldown period with a lower value for your scale in policy can help you reduce costs by allowing the group to scale in faster.

To create or update simple scaling policies in the console, choose the **Automatic scaling** tab after you create the group. To create or update simple scaling policies using the AWS CLI, use the put-scaling-policy command. For more information, see Step and simple scaling policies.

Scaling policy based on Amazon SQS

> **Important**
>
> The following information and steps shows you how to calculate the Amazon SQS queue backlog per instance using the `ApproximateNumberOfMessages` queue attribute before publishing it as a custom metric to CloudWatch. However, you can now save the cost and effort put into publishing your own metric by using metric math. For more information, see Create a target tracking scaling policy using metric math.

This section shows you how to scale your Auto Scaling group in response to changes in system load in an Amazon Simple Queue Service (Amazon SQS) queue. To learn more about how you can use Amazon SQS, see the Amazon Simple Queue Service Developer Guide.

There are some scenarios where you might think about scaling in response to activity in an Amazon SQS queue. For example, suppose that you have a web app that lets users upload images and use them online. In this scenario, each image requires resizing and encoding before it can be published. The app runs on EC2 instances in an Auto Scaling group, and it's configured to handle your typical upload rates. Unhealthy instances are terminated and replaced to maintain current instance levels at all times. The app places the raw bitmap data of the images in an SQS queue for processing. It processes the images and then publishes the processed images where they can be viewed by users. The architecture for this scenario works well if the number of image uploads doesn't vary over time. But if the number of uploads changes over time, you might consider using dynamic scaling to scale the capacity of your Auto Scaling group.

Contents

- Use target tracking with the right metric
- Limitations and prerequisites
- Configure scaling based on Amazon SQS
- Amazon SQS and instance scale-in protection

Use target tracking with the right metric

If you use a target tracking scaling policy based on a custom Amazon SQS queue metric, dynamic scaling can adjust to the demand curve of your application more effectively. For more information about choosing metrics for target tracking, see Choose metrics.

The issue with using a CloudWatch Amazon SQS metric like `ApproximateNumberOfMessagesVisible` for target tracking is that the number of messages in the queue might not change proportionally to the size of the Auto Scaling group that processes messages from the queue. That's because the number of messages in your SQS queue does not solely define the number of instances needed. The number of instances in your Auto Scaling group can be driven by multiple factors, including how long it takes to process a message and the acceptable amount of latency (queue delay).

The solution is to use a *backlog per instance* metric with the target value being the *acceptable backlog per instance* to maintain. You can calculate these numbers as follows:

- **Backlog per instance**: To calculate your backlog per instance, start with the `ApproximateNumberOfMessages` queue attribute to determine the length of the SQS queue (number of messages available for retrieval from the queue). Divide that number by the fleet's running capacity, which for an Auto Scaling group is the number of instances in the `InService` state, to get the backlog per instance.
- **Acceptable backlog per instance**: To calculate your target value, first determine what your application can accept in terms of latency. Then, take the acceptable latency value and divide it by the average time that an EC2 instance takes to process a message.

As an example, let's say that you currently have an Auto Scaling group with 10 instances and the number of visible messages in the queue (`ApproximateNumberOfMessages`) is 1500. If the average processing time is 0.1 seconds for each message and the longest acceptable latency is 10 seconds, then the acceptable backlog per instance is 10 / 0.1, which equals 100 messages. This

means that 100 is the target value for your target tracking policy. When the backlog per instance reaches the target value, a scale-out event will happen. Because the backlog per instance is already 150 messages (1500 messages / 10 instances), your group scales out, and it scales out by five instances to maintain proportion to the target value.

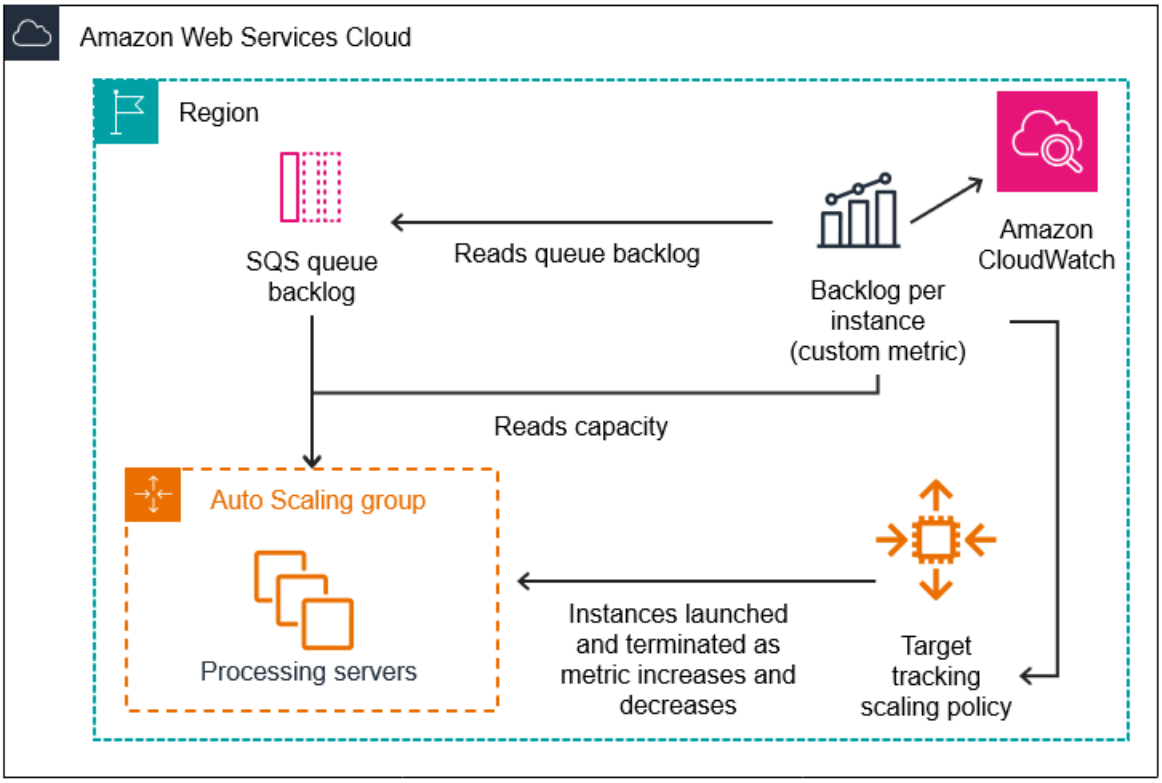
The following procedures demonstrate how to publish the custom metric and create the target tracking scaling policy that configures your Auto Scaling group to scale based on these calculations.

> **Important**
>
> Remember, to reduce costs, use metric math instead. For more information, see Create a target tracking scaling policy using metric math.

There are three main parts to this configuration:

- An Auto Scaling group to manage EC2 instances for the purposes of processing messages from an SQS queue.
- A custom metric to send to Amazon CloudWatch that measures the number of messages in the queue per EC2 instance in the Auto Scaling group.
- A target tracking policy that configures your Auto Scaling group to scale based on the custom metric and a set target value. CloudWatch alarms invoke the scaling policy.

The following diagram illustrates the architecture of this configuration.



Limitations and prerequisites

To use this configuration, you need to be aware of the following limitations:

- You must use the AWS CLI or an SDK to publish your custom metric to CloudWatch. You can then monitor your metric with the AWS Management Console.
- The Amazon EC2 Auto Scaling console does not support target tracking scaling policies that use custom metrics. You must use the AWS CLI or an SDK to specify a custom metric for your scaling policy.

The following sections direct you to use the AWS CLI for the tasks you need to perform. For example, to get metric data that reflects the present use of the queue, you use the SQS get-queue-attributes command. Make sure that you have the CLI installed and configured.

Before you begin, you must have an Amazon SQS queue to use. The following sections assume that you already have a queue (standard or FIFO), an Auto Scaling group, and EC2 instances running the application that uses the queue. For more information about Amazon SQS, see the Amazon Simple Queue Service Developer Guide.

Configure scaling based on Amazon SQS

This section describes how to configure your scaling based on Amazon Amazon SQS.

Tasks

- Step 1: Create a CloudWatch custom metric
- Step 2: Create a target tracking scaling policy
- Step 3: Test your scaling policy

Step 1: Create a CloudWatch custom metric

A custom metric is defined using a metric name and namespace of your choosing. Namespaces for custom metrics cannot start with `AWS/`. For more information about publishing custom metrics, see the Publish custom metrics topic in the *Amazon CloudWatch User Guide*.

Follow this procedure to create the custom metric by first reading information from your AWS account. Then, calculate the backlog per instance metric, as recommended in an earlier section. Lastly, publish this number to CloudWatch at a 1-minute granularity. Whenever possible, we strongly recommend that you scale on metrics with a 1-minute granularity to ensure a faster response to changes in system load.

To create a CloudWatch custom metric (AWS CLI)

1. Use the SQS get-queue-attributes command to get the number of messages waiting in the queue (`ApproximateNumberOfMessages`).

   ```
   aws sqs get-queue-attributes --queue-url https://
   sqs.region.amazonaws.com/123456789/MyQueue \
     --attribute-names ApproximateNumberOfMessages
   ```

2. Use the describe-auto-scaling-groups command to get the running capacity of the group, which is the number of instances in the `InService` lifecycle state. This command returns the instances of an Auto Scaling group along with their lifecycle state.

   ```
   aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names my-asg
   ```

3. Calculate the backlog per instance by dividing the approximate number of messages available for retrieval from the queue by the group's running capacity.

4. Create a script that runs every minute to retrieve the backlog per instance value and publish it to a CloudWatch custom metric. When you publish a custom metric, you specify the metric's name, namespace, unit, value, and zero or more dimensions. A dimension consists of a dimension name and a dimension value.

   To publish your custom metric, replace placeholder values in *italics* with your preferred metric name, the metric's value, a namespace (as long as it doesn't begin with "AWS"), and dimensions (optional), and then run the following put-metric-data command.

```
aws cloudwatch put-metric-data --metric-name MyBacklogPerInstance --
namespace MyNamespace \
  --unit None --value 20 --
dimensions MyOptionalMetricDimensionName=MyOptionalMetricDimensionValue
```

After your application is emitting the desired metric, the data is sent to CloudWatch. The metric is visible in the CloudWatch console. You can access it by logging into the AWS Management Console and navigating to the CloudWatch page. Then, view the metric by navigating to the metrics page or by searching for it using the search box. For information about viewing metrics, see View available metrics in the *Amazon CloudWatch User Guide*.

Step 2: Create a target tracking scaling policy

The metric you created can now be added to a target tracking scaling policy.

To create a target tracking scaling policy (AWS CLI)

1. Use the following `cat` command to store a target value for your scaling policy and a customized metric specification in a JSON file named `config.json` in your home directory. Replace each *`user input placeholder`* with your own information. For the `TargetValue`, calculate the acceptable backlog per instance metric and enter it here. To calculate this number, decide on a normal latency value and divide it by the average time that it takes to process a message, as described in an earlier section.

   If you didn't specify any dimensions for the metric you created in step 1, don't include any dimensions in the customized metric specification.

```
$ cat ~/config.json
{
   "TargetValue":100,
```

```
    "CustomizedMetricSpecification":{
       "MetricName":"MyBacklogPerInstance",
       "Namespace":"MyNamespace",
       "Dimensions":[
          {
             "Name":"MyOptionalMetricDimensionName",
             "Value":"MyOptionalMetricDimensionValue"
          }
       ],
       "Statistic":"Average",
       "Unit":"None"
    }
}
```

2. Use the put-scaling-policy command, along with the `config.json` file that you created in the previous step, to create your scaling policy.

```
aws autoscaling put-scaling-policy --policy-name sqs100-target-tracking-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://~/config.json
```

This creates two alarms: one for scaling out and one for scaling in. It also returns the Amazon Resource Name (ARN) of the policy that is registered with CloudWatch, which CloudWatch uses to invoke scaling whenever the metric threshold is in breach.

Step 3: Test your scaling policy

After your setup is complete, verify that your scaling policy is working. You can test it by increasing the number of messages in your SQS queue and then verifying that your Auto Scaling group has launched an additional EC2 instance. You can also test it by decreasing the number of messages in your SQS queue and then verifying that the Auto Scaling group has terminated an EC2 instance.

To test the scale-out function

1. Follow the steps in Creating an Amazon SQS standard queue and sending a message or Creating an Amazon SQS FIFO queue and sending a message to add messages to your queue. Make sure that you have increased the number of messages in the queue so that the backlog per instance metric exceeds the target value.

   It can take a few minutes for your changes to invoke the alarm.

2. Use the describe-auto-scaling-groups command to verify that the group has launched an instance.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

To test the scale in function

1. Follow the steps in Receive and delete a message (console) to delete messages from the queue. Make sure that you have decreased the number of messages in the queue so that the backlog per instance metric is below the target value.

   It can take a few minutes for your changes to invoke the alarm.

2. Use the describe-auto-scaling-groups command to verify that the group has terminated an instance.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

Amazon SQS and instance scale-in protection

Messages that have not been processed at the time an instance is terminated are returned to the SQS queue where they can be processed by another instance that is still running. For applications where long running tasks are performed, you can optionally use instance scale-in protection to have control over which queue workers are terminated when your Auto Scaling group scales in.

The following pseudocode shows one way to protect long-running, queue-driven worker processes from scale-in termination.

```
while (true)
{
  SetInstanceProtection(False);
  Work = GetNextWorkUnit();
  SetInstanceProtection(True);
  ProcessWorkUnit(Work);
  SetInstanceProtection(False);
}
```

For more information, see Design your applications to gracefully handle instance termination.

Verify a scaling activity for an Auto Scaling group

In the Amazon EC2 Auto Scaling section of the Amazon EC2 console, the **Activity history** for an Auto Scaling group lets you view the current status of a scaling activity that is currently in progress. When the scaling activity is finished, you can see whether it succeeds or not. This is particularly useful when you are creating Auto Scaling groups or you are adding scaling conditions to existing groups.

When you add a target tracking, step, or simple scaling policy to your Auto Scaling group, Amazon EC2 Auto Scaling immediately starts evaluating the policy against the metric. The metric alarm goes to ALARM state when the metric breaches the threshold for a specified number of evaluation periods. This means that a scaling policy could result in a scaling activity soon after it's created. After Amazon EC2 Auto Scaling adjusts the desired capacity in response to a scaling policy, you can verify the scaling activity in your account. If you want to receive email notification from Amazon EC2 Auto Scaling informing you about a scaling activity, follow the instructions in Amazon SNS notification options for Amazon EC2 Auto Scaling.

> **Tip**
>
> In the following procedure, you look at the **Activity history** and **Instances** sections for the Auto Scaling group. In both, the named columns should already be displayed. To display hidden columns or change the number of rows shown, choose the gear icon on the top right corner of each section to open the preferences modal, update the settings as needed, and choose **Confirm**.

To view the scaling activities for an Auto Scaling group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. In the navigation bar at the top of the screen, select the Region your Auto Scaling group is in.
3. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
4. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched or terminated instances, or whether the scaling activity is still in progress.

5. (Optional) If you have a lot of scaling activities, you can choose the **>** icon at the top edge of the activity history to see the next page of scaling activities.
6. On the **Instance management** tab, under **Instances**, the **Lifecycle** column contains the state of your instances. After the instance starts and any lifecycle hooks have finished, its lifecycle state changes to `InService`. The **Health status** column shows the result of the EC2 instance health check on your instance.

To view the scaling activities for an Auto Scaling group (AWS CLI)

Use the following describe-scaling-activities command.

```
aws autoscaling describe-scaling-activities --auto-scaling-group-name my-asg
```

The following is example output.

Scaling activities are ordered by start time. Activities still in progress are described first.

```
{
  "Activities": [
    {
      "ActivityId": "5e3a1f47-2309-415c-bfd8-35aa06300799",
      "AutoScalingGroupName": "my-asg",
      "Description": "Terminating EC2 instance: i-06c4794c2499af1df",
      "Cause": "At 2020-02-11T18:34:10Z a monitor alarm TargetTracking-my-asg-AlarmLow-
b9376cab-18a7-4385-920c-dfa3f7783f82 in state ALARM triggered policy my-target-
tracking-policy changing the desired capacity from 3 to 2.  At 2020-02-11T18:34:31Z
 an instance was taken out of service in response to a difference between desired and
 actual capacity, shrinking the capacity from 3 to 2.  At 2020-02-11T18:34:31Z instance
 i-06c4794c2499af1df was selected for termination.",
      "StartTime": "2020-02-11T18:34:31.268Z",
      "EndTime": "2020-02-11T18:34:53Z",
      "StatusCode": "Successful",
      "Progress": 100,
      "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":\"us-west-2a
\"...}",
      "AutoScalingGroupARN": "arn"
    },
...
  ]
}
```

For a description of the fields in the output, see Activity in the *Amazon EC2 Auto Scaling API Reference*.

For help retrieving the scaling activities for a deleted group and for information about the types of errors that you may encounter and how to handle them, see Troubleshoot issues in Amazon EC2 Auto Scaling.

Disable a scaling policy for an Auto Scaling group

This topic describes how to temporarily disable a scaling policy so it won't initiate changes to the number of instances the Auto Scaling group contains. When you disable a scaling policy, the configuration details are preserved, so you can quickly re-enable the policy. This is easier than temporarily deleting a policy when you don't need it, and recreating it later.

When a scaling policy is disabled, the Auto Scaling group does not scale out or scale in for the metric alarms that are breached while the scaling policy is disabled. However, any scaling activities still in progress are not stopped.

Note that disabled scaling policies still count toward your quotas on the number of scaling policies that you can add to an Auto Scaling group.

To disable a scaling policy (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Automatic scaling** tab, under **Dynamic scaling policies**, select the check box in the top right corner of the desired scaling policy.
4. Scroll to the top of the **Dynamic scaling policies** section, and choose **Actions**, **Disable**.

When you are ready to re-enable the scaling policy, repeat these steps and then choose **Actions**, **Enable**. After you re-enable a scaling policy, your Auto Scaling group may immediately initiate a scaling action if there are any alarms currently in ALARM state.

To disable a scaling policy (AWS CLI)

Use the put-scaling-policy command with the `--no-enabled` option as follows. Specify all options in the command as you would specify them when creating the policy.

```
aws autoscaling put-scaling-policy --auto-scaling-group-name my-asg \
   --policy-name my-scaling-policy --policy-type TargetTrackingScaling \
   --estimated-instance-warmup 360 \
   --target-tracking-configuration '{ "TargetValue": 70,
 "PredefinedMetricSpecification": { "PredefinedMetricType":
 "ASGAverageCPUUtilization" } }' \
   --no-enabled
```

To re-enable a scaling policy (AWS CLI)

Use the put-scaling-policy command with the `--enabled` option as follows. Specify all options in the command as you would specify them when creating the policy.

```
aws autoscaling put-scaling-policy --auto-scaling-group-name my-asg \
   --policy-name my-scaling-policy --policy-type TargetTrackingScaling \
   --estimated-instance-warmup 360 \
   --target-tracking-configuration '{ "TargetValue": 70,
 "PredefinedMetricSpecification": { "PredefinedMetricType":
 "ASGAverageCPUUtilization" } }' \
   --enabled
```

To describe a scaling policy (AWS CLI)

Use the describe-policies command to verify the enabled status of a scaling policy.

```
aws autoscaling describe-policies --auto-scaling-group-name my-asg \
   --policy-names my-scaling-policy
```

The following is example output.

```
{
    "ScalingPolicies": [
        {
            "AutoScalingGroupName": "my-asg",
            "PolicyName": "my-scaling-policy",
            "PolicyARN": "arn:aws:autoscaling:us-
west-2:123456789012:scalingPolicy:1d52783a-b03b-4710-
bb0e-549fd64378cc:autoScalingGroupName/my-asg:policyName/my-scaling-policy",
            "PolicyType": "TargetTrackingScaling",
```

```
            "StepAdjustments": [],
            "Alarms": [
                {
                    "AlarmName": "TargetTracking-my-asg-
AlarmHigh-9ca53fdd-7cf5-4223-938a-ae1199204502",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmHigh-9ca53fdd-7cf5-4223-938a-
ae1199204502"
                },
                {
                    "AlarmName": "TargetTracking-my-asg-AlarmLow-7010c83d-d55a-4a7a-
abe0-1cf8b9de6d6c",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-7010c83d-d55a-4a7a-
abe0-1cf8b9de6d6c"
                }
            ],
            "TargetTrackingConfiguration": {
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "ASGAverageCPUUtilization"
                },
                "TargetValue": 70.0,
                "DisableScaleIn": false
            },
            "Enabled": true
        }
    ]
}
```

Delete a scaling policy for an Auto Scaling group

After you no longer need a scaling policy, you can delete it. Depending on the type of scaling policy, you might also need to delete the CloudWatch alarms. Deleting a target tracking scaling policy also deletes any associated CloudWatch alarms. Deleting a step scaling policy or a simple scaling policy deletes the underlying alarm action, but it does not delete the CloudWatch alarm, even if it no longer has an associated action.

To delete a scaling policy (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

A split pane opens up in the bottom of the page.

3. On the **Automatic scaling** tab, under **Dynamic scaling policies**, select the check box in the top right corner of the desired scaling policy.
4. Scroll to the top of the **Dynamic scaling policies** section, and choose **Actions**, **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.
6. (Optional) If you deleted a step scaling policy or a simple scaling policy, do the following to delete the CloudWatch alarm that was associated with the policy. You can skip these substeps to keep the alarm for future use.

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

   b. On the navigation pane, choose **Alarms**.

   c. Choose the alarm (for example, `Step-Scaling-AlarmHigh-AddCapacity`) and choose **Action**, **Delete**.

   d. When prompted for confirmation, choose **Delete**.

To get the scaling policies for an Auto Scaling group (AWS CLI)

Before you delete a scaling policy, use the following describe-policies command to see what scaling policies were created for the Auto Scaling group. You can use the output when deleting the policy and the CloudWatch alarms.

```
aws autoscaling describe-policies --auto-scaling-group-name my-asg
```

You can filter the results by the type of scaling policy using the `--query` parameter. This syntax for `query` works on Linux or macOS. On Windows, change the single quotes to double quotes.

```
aws autoscaling describe-policies --auto-scaling-group-name my-asg
  --query 'ScalingPolicies[?PolicyType==`TargetTrackingScaling`]'
```

The following is example output.

```
[
    {
        "AutoScalingGroupName": "my-asg",
        "PolicyName": "cpu50-target-tracking-scaling-policy",
```

```
        "PolicyARN": "PolicyARN",
        "PolicyType": "TargetTrackingScaling",
        "StepAdjustments": [],
        "Alarms": [
            {
                "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e",
                "AlarmName": "TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e"
            },
            {
                "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2",
                "AlarmName": "TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2"
            }
        ],
        "TargetTrackingConfiguration": {
            "PredefinedMetricSpecification": {
                "PredefinedMetricType": "ASGAverageCPUUtilization"
            },
            "TargetValue": 50.0,
            "DisableScaleIn": false
        },
        "Enabled": true
    }
]
```

To delete your scaling policy (AWS CLI)

Use the following delete-policy command.

```
aws autoscaling delete-policy --auto-scaling-group-name my-asg \
  --policy-name cpu50-target-tracking-scaling-policy
```

To delete your CloudWatch alarm (AWS CLI)

For step and simple scaling policies, use the delete-alarms command to delete the CloudWatch alarm that was associated with the policy. You can skip this step to keep the alarm for future use. You can delete one or more alarms at a time. For example, use the following command

to delete the `Step-Scaling-AlarmHigh-AddCapacity` and `Step-Scaling-AlarmLow-RemoveCapacity` alarms.

```
aws cloudwatch delete-alarms --alarm-name Step-Scaling-AlarmHigh-AddCapacity Step-Scaling-AlarmLow-RemoveCapacity
```

Example scaling policies for the AWS CLI

You can create scaling policies for Amazon EC2 Auto Scaling through the AWS Management Console, AWS Command Line Interface (AWS CLI), or SDKs.

The following examples show how you can create scaling policies for Amazon EC2 Auto Scaling with the AWS CLI put-scaling-policy command. Replace each *`user input placeholder`* with your own information.

To get started with writing scaling policies using the AWS CLI, see the introductory exercises in Target tracking scaling policies and Step and simple scaling policies.

Example 1: To apply a target tracking scaling policy with a predefined metric specification

```
aws autoscaling put-scaling-policy --policy-name cpu50-target-tracking-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://config.json
{
  "TargetValue": 50.0,
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "ASGAverageCPUUtilization"
  }
}
```

For more information, see PredefinedMetricSpecification in the *Amazon EC2 Auto Scaling API Reference*.

> **Note**
>
> If the file is not in the current directory, type the full path to file. For more information about reading AWS CLI parameter values from a file, see Loading AWS CLI parameters from a file in the AWS Command Line Interface User Guide.

Example 2: To apply a target tracking scaling policy with a customized metric specification

```
aws autoscaling put-scaling-policy --policy-name sqs100-target-tracking-scaling-policy
 \
  --auto-scaling-group-name my-asg --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://config.json
{
  "TargetValue": 100.0,
  "CustomizedMetricSpecification": {
    "MetricName": "MyBacklogPerInstance",
    "Namespace": "MyNamespace",
    "Dimensions": [{
      "Name": "MyOptionalMetricDimensionName",
      "Value": "MyOptionalMetricDimensionValue"
    }],
    "Statistic": "Average",
    "Unit": "None"
  }
}
```

For more information, see CustomizedMetricSpecification in the *Amazon EC2 Auto Scaling API Reference*.

Example 3: To apply a target tracking scaling policy for scale out only

```
aws autoscaling put-scaling-policy --policy-name alb1000-target-tracking-scaling-policy
 \
  --auto-scaling-group-name my-asg --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://config.json
{
  "TargetValue": 1000.0,
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "ALBRequestCountPerTarget",
    "ResourceLabel": "app/my-alb/778d41231b141a0f/targetgroup/my-alb-target-
group/943f017f100becff"
  },
  "DisableScaleIn": true
}
```

Example 4: To apply a step scaling policy for scale out

```
aws autoscaling put-scaling-policy \
```

```
  --auto-scaling-group-name my-asg  \
  --policy-name my-step-scale-out-policy \
  --policy-type StepScaling \
  --adjustment-type PercentChangeInCapacity \
  --metric-aggregation-type Average \
  --step-adjustments
 MetricIntervalLowerBound=10.0,MetricIntervalUpperBound=20.0,ScalingAdjustment=10 \

 MetricIntervalLowerBound=20.0,MetricIntervalUpperBound=30.0,ScalingAdjustment=20 \
                     MetricIntervalLowerBound=30.0,ScalingAdjustment=30 \
  --min-adjustment-magnitude 1
```

Record the policy's Amazon Resource Name (ARN). You need the ARN when you create the CloudWatch alarm.

Example 5: To apply a step scaling policy for scale in

```
aws autoscaling put-scaling-policy \
  --auto-scaling-group-name my-asg  \
  --policy-name my-step-scale-in-policy \
  --policy-type StepScaling \
  --adjustment-type ChangeInCapacity \
  --step-adjustments MetricIntervalUpperBound=0.0,ScalingAdjustment=-2
```

Record the policy's Amazon Resource Name (ARN). You need the ARN when you create the CloudWatch alarm.

Example 6: To apply a simple scaling policy for scale out

```
aws autoscaling put-scaling-policy --policy-name my-simple-scale-out-policy \
  --auto-scaling-group-name my-asg --scaling-adjustment 30 \
  --adjustment-type PercentChangeInCapacity --min-adjustment-magnitude 2
```

Record the policy's Amazon Resource Name (ARN). You need the ARN when you create the CloudWatch alarm.

Example 7: To apply a simple scaling policy for scale in

```
aws autoscaling put-scaling-policy --policy-name my-simple-scale-in-policy \
  --auto-scaling-group-name my-asg --scaling-adjustment -1 \
  --adjustment-type ChangeInCapacity --cooldown 180
```

Record the policy's Amazon Resource Name (ARN). You need the ARN when you create the CloudWatch alarm.

Predictive scaling for Amazon EC2 Auto Scaling

Predictive scaling works by analyzing historical load data to detect daily or weekly patterns in traffic flows. It uses this information to forecast future capacity needs so Amazon EC2 Auto Scaling can proactively increase the capacity of your Auto Scaling group to match the anticipated load.

Predictive scaling is well suited for situations where you have:

- Cyclical traffic, such as high use of resources during regular business hours and low use of resources during evenings and weekends
- Recurring on-and-off workload patterns, such as batch processing, testing, or periodic data analysis
- Applications that take a long time to initialize, causing a noticeable latency impact on application performance during scale-out events

In general, if you have regular patterns of traffic increases and applications that take a long time to initialize, you should consider using predictive scaling. Predictive scaling can help you scale faster by launching capacity in advance of forecasted load, compared to using only dynamic scaling, which is reactive in nature. Predictive scaling can also potentially save you money on your EC2 bill by helping you avoid the need to over provision capacity.

For example, consider an application that has high usage during business hours and low usage overnight. At the start of each business day, predictive scaling can add capacity before the first influx of traffic. This helps your application maintain high availability and performance when going from a period of lower utilization to a period of higher utilization. You don't have to wait for dynamic scaling to react to changing traffic. You also don't have to spend time reviewing your application's load patterns and trying to schedule the right amount of capacity using scheduled scaling.

Topics

- How predictive scaling works
- Create a predictive scaling policy for an Auto Scaling group
- Evaluate your predictive scaling policies

- Override forecast values using scheduled actions
- Advanced predictive scaling policy using custom metrics

How predictive scaling works

This topic explains how predictive scaling works and describes what to consider when you create a predictive scaling policy.

Topics

- How it works
- Maximum capacity limit
- Considerations
- Supported Regions

How it works

To use predictive scaling, create a predictive scaling policy that specifies the CloudWatch metric to monitor and analyze. For predictive scaling to start forecasting future values, this metric must have at least 24 hours of data.

After you create the policy, predictive scaling starts analyzing metric data from up to the past 14 days to identify patterns. It uses this analysis to generate an hourly forecast of capacity requirements for the next 48 hours. The forecast is updated every 6 hours using the latest CloudWatch data. As new data comes in, predictive scaling is able to continuously improve the accuracy of future forecasts.

When you first enable predictive scaling, it runs in *forecast only* mode. In this mode, it generates capacity forecasts but does not actually scale your Auto Scaling group based on those forecasts. This allows you to evaluate the accuracy and suitability of the forecast. You can view forecast data by using the `GetPredictiveScalingForecast` API operation or the AWS Management Console.

After you review the forecast data and decide to start scaling based on that data, switch the scaling policy to *forecast and scale* mode. In this mode:

- If the forecast expects an increase in load, Amazon EC2 Auto Scaling will increase capacity by scaling out.

- If the forecast expects a decrease in load, it will not scale in to remove capacity. If you want to remove capacity that is no longer needed, you must create dynamic scaling policies.

By default, Amazon EC2 Auto Scaling scales your Auto Scaling group at the start of each hour based on the forecast for that hour. You can optionally specify an earlier start time by using the `SchedulingBufferTime` property in the `PutScalingPolicy` API operation or the **Pre-launch instances** setting in the AWS Management Console. This causes Amazon EC2 Auto Scaling to launch new instances ahead of the forecasted demand, giving them time to boot and become ready to handle traffic.

To support launching new instances ahead of the forecasted demand, we strongly recommend that you enable the *default instance warmup* for your Auto Scaling group. This specifies a time period after a scale-out activity during which Amazon EC2 Auto Scaling won't scale in, even if dynamic scaling policies indicate capacity should be decreased. This helps you ensure that newly launched instances have adequate time to start serving the increased traffic before being considered for scale-in operations. For more information, see Set the default instance warmup for an Auto Scaling group.

Maximum capacity limit

Auto Scaling groups have a maximum capacity setting that limits the maximum number of EC2 instances that can be launched for the group. By default, when scaling policies are set, they cannot increase capacity higher than its maximum capacity.

Alternatively, you can allow the group's maximum capacity to be automatically increased if the forecast capacity approaches or exceeds the maximum capacity of the Auto Scaling group. To enable this behavior, use the `MaxCapacityBreachBehavior` and `MaxCapacityBuffer` properties in the `PutScalingPolicy` API operation or the **Max capacity behavior** setting in the AWS Management Console.

> **Warning**
>
> Use caution when allowing the maximum capacity to be automatically increased. This can lead to more instances being launched than intended if the increased maximum capacity is not monitored and managed. The increased maximum capacity then becomes the new normal maximum capacity for the Auto Scaling group until you manually update it. The maximum capacity does not automatically decrease back to the original maximum.

Considerations

- Confirm whether predictive scaling is suitable for your workload. A workload is a good fit for predictive scaling if it exhibits recurring load patterns that are specific to the day of the week or the time of day. To check this, configure predictive scaling policies in *forecast only* mode and then refer to the recommendations in the console. Amazon EC2 Auto Scaling provides recommendations based on observations about potential policy performance. Evaluate the forecast and the recommendations before letting predictive scaling actively scale your application.
- Predictive scaling needs at least 24 hours of historical data to start forecasting. However, forecasts are more effective if historical data spans two full weeks. If you update your application by creating a new Auto Scaling group and deleting the old one, then your new Auto Scaling group needs 24 hours of historical load data before predictive scaling can start generating forecasts again. You can use custom metrics to aggregate metrics across old and new Auto Scaling groups. Otherwise, you might have to wait a few days for a more accurate forecast.
- Choose a load metric that accurately represents the full load on your application and is the aspect of your application that's most important to scale on.
- Using dynamic scaling with predictive scaling helps you follow the demand curve for your application closely, scaling in during periods of low traffic and scaling out when traffic is higher than expected. When multiple scaling policies are active, each policy determines the desired capacity independently, and the desired capacity is set to the maximum of those. For example, if 10 instances are required to stay at the target utilization in a target tracking scaling policy, and 8 instances are required to stay at the target utilization in a predictive scaling policy, then the group's desired capacity is set to 10. If you are new to dynamic scaling, we recommend using target tracking scaling policies. For more information, see Dynamic scaling for Amazon EC2 Auto Scaling.
- A core assumption of predictive scaling is that the Auto Scaling group is homogenous and all instances are of equal capacity. If this isn't true for your group, forecasted capacity can be inaccurate. Therefore, use caution when creating predictive scaling policies for mixed instances groups because instances of different types can be provisioned that are of unequal capacity. Following are some examples where the forecasted capacity will be inaccurate:
  - Your predictive scaling policy is based on CPU utilization, but the number of vCPUs on each Auto Scaling instance varies between instance types.
  - Your predictive scaling policy is based on network in or network out, but the network bandwidth throughput for each Auto Scaling instance varies between instance types. For

example, the M5 and M5n instance types are similar, but the M5n instance type delivers significantly higher network throughput.

Supported Regions

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Jakarta)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)

- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Create a predictive scaling policy for an Auto Scaling group

The following procedures help you create a predictive scaling policy using the AWS Management Console or AWS CLI.

If the Auto Scaling group is new, it must provide at least 24 hours of data before Amazon EC2 Auto Scaling can generate a forecast for it.

Contents

- Create a predictive scaling policy (console)
- Create a predictive scaling policy (AWS CLI)

Create a predictive scaling policy (console)

If this is your first time creating a predictive scaling policy, we recommend using the console to create multiple predictive scaling policies in *forecast only* mode. This allows you to test the potential effects of different metrics and target values. You can create multiple predictive scaling policies for each Auto Scaling group, but only one of the policies can be used for active scaling.

Create a predictive scaling policy in the console (predefined metrics)

Use the following procedure to create a predictive scaling policy using predefined metrics (CPU, network I/O, or Application Load Balancer request count per target). The easiest way to create a predictive scaling policy is to use predefined metrics. If you prefer to use custom metrics instead, see Create a predictive scaling policy in the console (custom metrics).

To create a predictive scaling policy

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up at the bottom of the page.
3. On the **Automatic scaling** tab, in **Scaling policies**, choose **Create predictive scaling policy**.

4. Enter a name for the policy.
5. Turn on **Scale based on forecast** to give Amazon EC2 Auto Scaling permission to start scaling right away.

   To keep the policy in *forecast only* mode, keep **Scale based on forecast** turned off.
6. For **Metrics**, choose your metrics from the list of options. Options include **CPU**, **Network In**, **Network Out**, **Application Load Balancer request count**, and **Custom metric pair**.

   If you chose **Application Load Balancer request count per target**, then choose a target group in **Target group**. **Application Load Balancer request count per target** is only supported if you have attached an Application Load Balancer target group to your Auto Scaling group.

   If you chose **Custom metric pair**, choose individual metrics from the drop-down lists for **Load metric** and **Scaling metric**.
7. For **Target utilization**, enter the target value that Amazon EC2 Auto Scaling should maintain. Amazon EC2 Auto Scaling scales out your capacity until the average utilization is at the target utilization, or until it reaches the maximum number of instances you specified.

If your scaling metric is...	Then the target utilization represents...
CPU	The percentage of CPU that each instance should ideally use.
Network In	The average number of bytes per minute that each instance should ideally receive.
Network Out	The average number of bytes per minute that each instance should ideally send out.
Application Load Balancer request count per target	The average number of requests per minute that each instance should ideally receive.

8. (Optional) For **Pre-launch instances**, choose how far in advance you want your instances launched before the forecast calls for the load to increase.
9. (Optional) For **Max capacity behavior**, choose whether to let Amazon EC2 Auto Scaling scale out higher than the group's maximum capacity when predicted capacity exceeds the defined maximum. Turning on this setting lets scale out occur during periods when your traffic is forecasted to be at its highest.

10. (Optional) For **Buffer maximum capacity above the forecasted capacity**, choose how much additional capacity to use when the predicted capacity is close to or exceeds the maximum capacity. The value is specified as a percentage relative to the predicted capacity. For example, if the buffer is 10, this means a 10 percent buffer. Therefore, if the predicted capacity is 50 and the maximum capacity is 40, the effective maximum capacity is 55.

    If set to 0, Amazon EC2 Auto Scaling might scale capacity higher than the maximum capacity to equal but not exceed predicted capacity.
11. Choose **Create predictive scaling policy**.

Create a predictive scaling policy in the console (custom metrics)

Use the following procedure to create a predictive scaling policy using custom metrics. Custom metrics can include other metrics provided by CloudWatch or metrics that you publish to CloudWatch. To use CPU, network I/O, or Application Load Balancer request count per target, see Create a predictive scaling policy in the console (predefined metrics).

To create a predictive scaling policy using custom metrics, you must do the following:

- You must provide the raw queries that let Amazon EC2 Auto Scaling interact with the metrics in CloudWatch. For more information, see Advanced predictive scaling policy using custom metrics. To be sure that Amazon EC2 Auto Scaling can extract the metric data from CloudWatch, confirm that each query is returning data points. Confirm this by using the CloudWatch console or the CloudWatch GetMetricData API operation.

  > **Note**
  >
  > We provide sample JSON payloads in the JSON editor in the Amazon EC2 Auto Scaling console. These examples give you a reference for the key-value pairs that are required to add other CloudWatch metrics provided by AWS or metrics that you previously published to CloudWatch. You can use them as a starting point, then customize them for your needs.

- If you use any metric math, you must manually construct the JSON to fit your unique scenario. For more information, see Use metric math expressions. Before using metric math in your policy, confirm that metric queries based on metric math expressions are valid and return a single time series. Confirm this by using the CloudWatch console or the CloudWatch GetMetricData API operation.

If you make an error in a query by providing incorrect data, such as the wrong Auto Scaling group name, the forecast won't have any data. For troubleshooting custom metric issues, see Considerations for custom metrics in a predictive scaling policy.

To create a predictive scaling policy

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up at the bottom of the page.
3. On the **Automatic scaling** tab, in **Scaling policies**, choose **Create predictive scaling policy**.
4. Enter a name for the policy.
5. Turn on **Scale based on forecast** to give Amazon EC2 Auto Scaling permission to start scaling right away.

   To keep the policy in *forecast only* mode, keep **Scale based on forecast** turned off.
6. For **Metrics**, choose **Custom metric pair**.

   a. For **Load metric**, choose **Custom CloudWatch metric** to use a custom metric. Construct the JSON payload that contains the load metric definition for the policy and paste it into the JSON editor box, replacing what is already in the box.

   b. For **Scaling metric**, choose **Custom CloudWatch metric** to use a custom metric. Construct the JSON payload that contains the scaling metric definition for the policy and paste it into the JSON editor box, replacing what is already in the box.

   c. (Optional) To add a custom capacity metric, select the check box for **Add custom capacity metric**. Construct the JSON payload that contains the capacity metric definition for the policy and paste it into the JSON editor box, replacing what is already in the box.

      You only need to enable this option to create a new time series for capacity if your capacity metric data spans multiple Auto Scaling groups. In this case, you must use metric math to aggregate the data into a single time series.
7. For **Target utilization**, enter the target value that Amazon EC2 Auto Scaling should maintain. Amazon EC2 Auto Scaling scales out your capacity until the average utilization is at the target utilization, or until it reaches the maximum number of instances you specified.
8. (Optional) For **Pre-launch instances**, choose how far in advance you want your instances launched before the forecast calls for the load to increase.

9. (Optional) For **Max capacity behavior**, choose whether to let Amazon EC2 Auto Scaling scale out higher than the group's maximum capacity when predicted capacity exceeds the defined maximum. Turning on this setting lets scale out occur during periods when your traffic is forecasted to be at its highest.
10. (Optional) For **Buffer maximum capacity above the forecasted capacity**, choose how much additional capacity to use when the predicted capacity is close to or exceeds the maximum capacity. The value is specified as a percentage relative to the predicted capacity. For example, if the buffer is 10, this means a 10 percent buffer. Therefore, if the predicted capacity is 50 and the maximum capacity is 40, the effective maximum capacity is 55.

    If set to 0, Amazon EC2 Auto Scaling might scale capacity higher than the maximum capacity to equal but not exceed predicted capacity.
11. Choose **Create predictive scaling policy**.

Create a predictive scaling policy (AWS CLI)

Use the AWS CLI as follows to configure predictive scaling policies for your Auto Scaling group. Replace each *`user input placeholder`* with your own information.

For more information about the CloudWatch metrics you can specify, see PredictiveScalingMetricSpecification in the *Amazon EC2 Auto Scaling API Reference*.

Example 1: A predictive scaling policy that creates forecasts but doesn't scale

The following example policy shows a complete policy configuration that uses CPU utilization metrics for predictive scaling with a target utilization of `40`. `ForecastOnly` mode is used by default, unless you explicitly specify which mode to use. Save this configuration in a file named `config.json`.

```
{
    "MetricSpecifications": [
        {
            "TargetValue": 40,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ASGCPUUtilization"
            }
        }
    ]
}
```

To create the policy from the command line, run the put-scaling-policy command with the configuration file specified, as demonstrated in the following example.

```
aws autoscaling put-scaling-policy --policy-name cpu40-predictive-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN).

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/cpu40-predictive-scaling-
policy",
  "Alarms": []
}
```

Example 2: A predictive scaling policy that forecasts and scales

For a policy that allows Amazon EC2 Auto Scaling to forecast and scale, add the property `Mode` with a value of `ForecastAndScale`. The following example shows a policy configuration that uses Application Load Balancer request count metrics. The target utilization is `1000`, and predictive scaling is set to `ForecastAndScale` mode.

```
{
    "MetricSpecifications": [
        {
            "TargetValue": 1000,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ALBRequestCount",
                "ResourceLabel": "app/my-alb/778d41231b141a0f/targetgroup/my-alb-
target-group/943f017f100becff"
            }
        }
    ],
    "Mode": "ForecastAndScale"
}
```

To create this policy, run the put-scaling-policy command with the configuration file specified, as demonstrated in the following example.

```
aws autoscaling put-scaling-policy --policy-name alb1000-predictive-scaling-policy \
```

```
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN).

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-
id:scalingPolicy:19556d63-7914-4997-8c81-d27ca5241386:autoScalingGroupName/my-
asg:policyName/alb1000-predictive-scaling-policy",
  "Alarms": []
}
```

Example 3: A predictive scaling policy that can scale higher than maximum capacity

The following example shows how to create a policy that can scale higher than the group's maximum size limit when you need it to handle a higher than normal load. By default, Amazon EC2 Auto Scaling doesn't scale your EC2 capacity higher than your defined maximum capacity. However, it might be helpful to let it scale higher with slightly more capacity to avoid performance or availability issues.

To provide room for Amazon EC2 Auto Scaling to provision additional capacity when the capacity is predicted to be at or very close to your group's maximum size, specify the `MaxCapacityBreachBehavior` and `MaxCapacityBuffer` properties, as shown in the following example. You must specify `MaxCapacityBreachBehavior` with a value of `IncreaseMaxCapacity`. The maximum number of instances that your group can have depends on the value of `MaxCapacityBuffer`.

```
{
    "MetricSpecifications": [
        {
            "TargetValue": 70,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ASGCPUUtilization"
            }
        }
    ],
    "MaxCapacityBreachBehavior": "IncreaseMaxCapacity",
    "MaxCapacityBuffer": 10
}
```

In this example, the policy is configured to use a 10 percent buffer (`"MaxCapacityBuffer": 10`), so if the predicted capacity is 50 and the maximum capacity is 40, then the effective maximum capacity is 55. A policy that can scale capacity higher than the maximum capacity to equal but not exceed predicted capacity would have a buffer of 0 (`"MaxCapacityBuffer": 0`).

To create this policy, run the put-scaling-policy command with the configuration file specified, as demonstrated in the following example.

```
aws autoscaling put-scaling-policy --policy-name cpu70-predictive-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN).

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:d02ef525-8651-4314-
bf14-888331ebd04f:autoScalingGroupName/my-asg:policyName/cpu70-predictive-scaling-
policy",
  "Alarms": []
}
```

Evaluate your predictive scaling policies

Before you use a predictive scaling policy to scale your Auto Scaling group, review the recommendations and other data for your policy in the Amazon EC2 Auto Scaling console. This is important because you don't want a predictive scaling policy to scale your actual capacity until you know that its predictions are accurate.

If the Auto Scaling group is new, give Amazon EC2 Auto Scaling 24 hours to create the first forecast.

When Amazon EC2 Auto Scaling creates a forecast, it uses historical data. If your Auto Scaling group doesn't have much recent historical data yet, Amazon EC2 Auto Scaling might temporarily backfill the forecast with aggregates created from the currently available historical aggregates. Forecasts are backfilled for up to two weeks before a policy's creation date.

Contents

- View your predictive scaling recommendations
- Review predictive scaling monitoring graphs

- Monitor predictive scaling metrics with CloudWatch

View your predictive scaling recommendations

For effective analysis, Amazon EC2 Auto Scaling should have at least two predictive scaling policies to compare. (However, you can still review the findings for a single policy.) When you create multiple policies, you can evaluate a policy that uses one metric against a policy that uses a different metric. You can also evaluate the impact of different target value and metric combinations. After the predictive scaling policies are created, Amazon EC2 Auto Scaling immediately starts evaluating which policy would do a better job of scaling your group.

To view your recommendations in the Amazon EC2 Auto Scaling console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Auto scaling** tab, under **Predictive scaling policies**, you can view details about a policy along with our recommendation. The recommendation tells you whether the predictive scaling policy does a better job than not using it.

   If you're unsure whether a predictive scaling policy is appropriate for your group, review the **Availability impact** and **Cost impact** columns to choose the right policy. The information for each column tells you what the impact of the policy is.

   - **Availability impact**: Describes whether the policy would avoid negative impact to availability by provisioning enough instances to handle the workload, compared to not using the policy.
   - **Cost impact**: Describes whether the policy would avoid negative impact on your costs by not over-provisioning instances, compared to not using the policy. By over-provisioning too much, your instances are underutilized or idle, which only adds to the cost impact.

   If you have multiple policies, then a **Best prediction** tag displays next to the name of the policy that gives the most availability benefits at lower cost. More weight is given to availability impact.

4. (Optional) To select the desired time period for recommendation results, choose your preferred value from the **Evaluation period** dropdown: **2 days**, **1 week**, **2 weeks**, **4 weeks**, **6 weeks**, or **8 weeks**. By default, the evaluation period is the last two weeks. A longer evaluation period provides more data points to the recommendation results. However, adding more data points might not improve the results if your load patterns have changed, such as after a period of exceptional demand. In this case, you can get a more focused recommendation by looking at more recent data.

> **Note**
>
> Recommendations are generated only for policies that are in **Forecast only** mode. The recommendations feature works better when a policy is in the **Forecast only** mode throughout the evaluation period. If you start a policy in **Forecast and scale** mode and switch it to **Forecast only** mode later, the findings for that policy are likely to be biased. This is because the policy has already contributed toward the actual capacity.

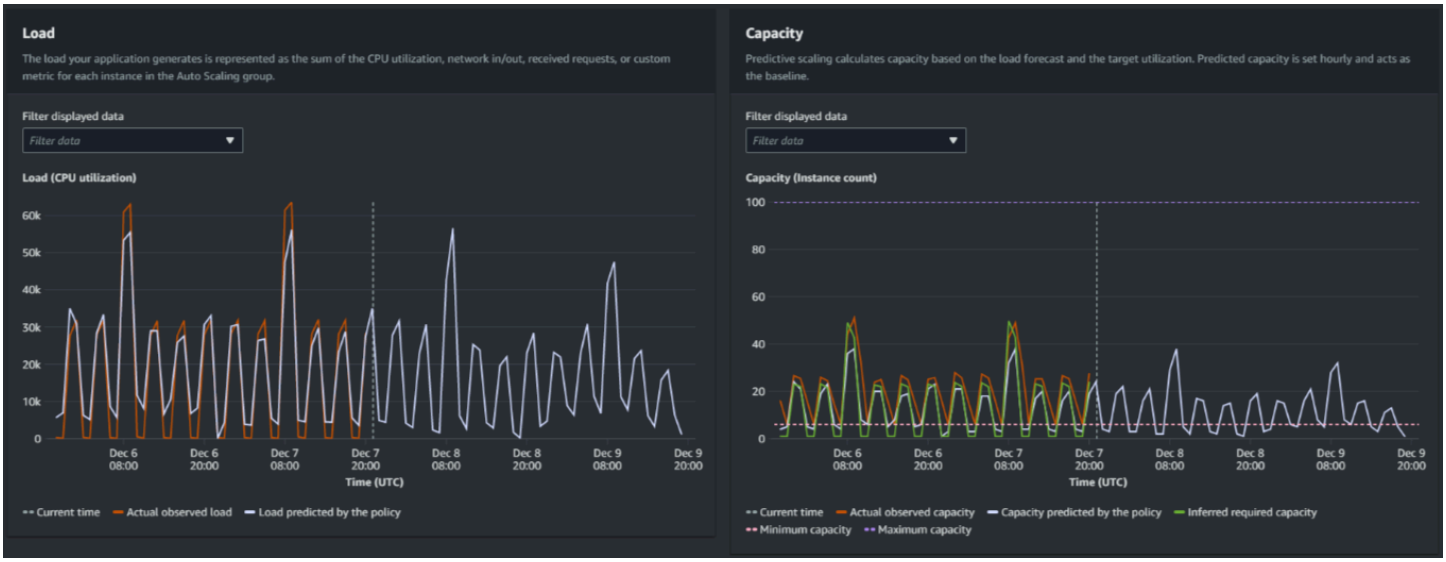
Review predictive scaling monitoring graphs

In the Amazon EC2 Auto Scaling console, you can review the forecast of the previous days, weeks, or months to visualize how well the policy performs over time. You can also use this information to evaluate the accuracy of predictions when deciding whether to let a policy scale your actual capacity.

To review predictive scaling monitoring graphs in the Amazon EC2 Auto Scaling console

1. Choose a policy from the **Predictive scaling policies** list.
2. In the **Monitoring** section, you can view your policy's past and future forecasts for load and capacity against actual values. The **Load** graph shows load forecast and actual values for the load metric that you chose. The **Capacity** graph shows the number of instances predicted by the policy. It also includes the actual number of instances launched. The vertical line separates historical values from future forecasts. These graphs become available shortly after the policy is created.
3. (Optional) To change the amount of historical data shown in the chart, choose your preferred value from the **Evaluation period** dropdown at the top of the page. The evaluation period does not transform the data on this page in any way. It only changes the amount of historical data shown.

The following image shows the **Load** and **Capacity** graphs when forecasts have been applied multiple times. Predictive scaling forecasts load based on your historical load data. The load your application generates is represented as the sum of the CPU utilization, network in/out, received requests, or custom metric for each instance in the Auto Scaling group. Predictive scaling calculates future capacity needs based on the load forecast and the target utilization that you want to achieve for the scaling metric.



Compare data in the Load graph

Each horizontal line represents a different set of data points reported in one-hour intervals:

1. **Actual observed load** uses the SUM statistic for your chosen load metric to show the total hourly load in the past.
2. **Load predicted by the policy** shows the hourly load prediction. This prediction is based on the previous two weeks of actual load observations.

Compare data in the Capacity graph

Each horizontal line represents a different set of data points reported in one-hour intervals:

1. **Actual observed capacity** shows your Auto Scaling group's actual capacity in the past, which depends on your other scaling policies and minimum group size in effect for the selected time period.
2. **Capacity predicted by the policy** shows the baseline capacity that you can expect to have at the beginning of each hour when the policy is in **Forecast and scale** mode.

3. **Inferred required capacity** shows the ideal capacity to maintain the scaling metric at the target value you chose.
4. **Minimum capacity** shows the minimum capacity of the Auto Scaling group.
5. **Maximum capacity** shows the maximum capacity of the Auto Scaling group.

For the purpose of calculating the inferred required capacity, we begin by assuming that each instance is equally utilized at a specified target value. In practice, instances are not equally utilized. By assuming that utilization is uniformly spread between instances, however, we can make a likelihood estimate of the amount of capacity that is needed. The capacity requirement is then calculated to be inversely proportional to the scaling metric that you used for your predictive scaling policy. In other words, as capacity increases, the scaling metric decreases at the same rate. For example, if capacity doubles, the scaling metric must decrease by half.

The formula for the inferred required capacity:

```
sum of (actualCapacityUnits*scalingMetricValue)/(targetUtilization)
```

For example, we take the `actualCapacityUnits` (`10`) and the `scalingMetricValue` (`30`) for a given hour. We then take the `targetUtilization` that you specified in your predictive scaling policy (`60`) and calculate the inferred required capacity for the same hour. This returns a value of 5. This means that five is the inferred amount of capacity required to maintain capacity in direct inverse proportion to the target value of the scaling metric.

> **Note**
>
> Various levers are available for you to adjust and improve the cost savings and availability of your application.
>
> - You use predictive scaling for the baseline capacity and dynamic scaling to handle additional capacity. Dynamic scaling works independently from predictive scaling, scaling in and out based on current utilization. First, Amazon EC2 Auto Scaling calculates the recommended number of instances for each dynamic scaling policy. Then, it scales based on the policy that provides the largest number of instances.
> - To allow scale in to occur when the load decreases, your Auto Scaling group should always have at least one dynamic scaling policy with the scale-in portion enabled.
> - You can improve scaling performance by making sure that your minimum and maximum capacity are not too restrictive. A policy with a recommended number of instances that

does not fall within the minimum and maximum capacity range will be prevented from scaling in and out.

Monitor predictive scaling metrics with CloudWatch

Depending on your needs, you might prefer to access monitoring data for predictive scaling from Amazon CloudWatch instead of the Amazon EC2 Auto Scaling console. After you create a predictive scaling policy, the policy collects data that is used to forecast your future load and capacity. After this data is collected, it is automatically stored in CloudWatch at regular intervals. Then, you can use CloudWatch to visualize how well the policy performs over time. You can also create CloudWatch alarms to notify you when performance indicators change beyond the limits that you define in CloudWatch.

Topics

- Visualize historical forecast data
- Create accuracy metrics using metric math

Visualize historical forecast data

You can view the load and capacity forecast data for a predictive scaling policy in CloudWatch. This can be useful when visualizing forecasts against other CloudWatch metrics in a single graph. It can also help when viewing a broader time range so that you can see trends over time. You can access up to 15 months of historical metrics to get a better perspective on how your policy is performing.

For more information, see Predictive scaling metrics and dimensions.

To view historical forecast data using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics** and then **All metrics**.
3. Choose the **Auto Scaling** metric namespace.
4. Choose one of the following options to view either the load forecast or capacity forecast metrics:
   - **Predictive Scaling Load Forecasts**
   - **Predictive Scaling Capacity Forecasts**

5. In the search field, enter the name of the predictive scaling policy or the name of the Auto Scaling group, and then press Enter to filter the results.
6. To graph a metric, select the check box next to the metric. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**. For more information, see Graphing a metric in the *Amazon CloudWatch User Guide*.
7. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value, and then choose a different statistic. Although you can choose any statistic for each metric, not all statistics are useful for **PredictiveScalingLoadForecast** and **PredictiveScalingCapacityForecast** metrics. For example, the **Average**, **Minimum**, and **Maximum** statistics are useful, but the **Sum** statistic is not.
8. To add another metric to the graph, under **Browse**, choose **All**, find the specific metric, and then select the check box next to it. You can add up to 10 metrics.

   For example, to add the actual values for CPU utilization to the graph, choose the **EC2** namespace and then choose **By Auto Scaling Group**. Then, select the check box for the **CPUUtilization** metric and the specific Auto Scaling group.
9. (Optional) To add the graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Create accuracy metrics using metric math

With metric math, you can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series on the CloudWatch console and add them to dashboards. For more information about metric math, see Using metric math in the *Amazon CloudWatch User Guide*.

Using metric math, you can graph the data that Amazon EC2 Auto Scaling generates for predictive scaling in different ways. This helps you monitor policy performance over time, and helps you understand whether your combination of metrics can be improved.

For example, you can use a metric math expression to monitor the mean absolute percentage error (MAPE). The MAPE metric helps monitor the difference between the forecasted values and the actual values observed during a given forecast window. Changes in the value of MAPE can indicate whether the policy's performance is degrading over time as the nature of your application changes. An increase in MAPE signals a wider gap between the forecasted values and the actual values.

Example: Metric math expression

To get started with this type of graph, you can create a metric math expression like the one shown in the following example.

```
{
  "MetricDataQueries": [
    {
      "Expression": "TIME_SERIES(AVG(ABS(m1-m2)/m1))",
      "Id": "e1",
      "Period": 3600,
      "Label": "MeanAbsolutePercentageError",
      "ReturnData": true
    },
    {
      "Id": "m1",
      "Label": "ActualLoadValues",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/EC2",
          "MetricName": "CPUUtilization",
          "Dimensions": [
            {
              "Name": "AutoScalingGroupName",
              "Value": "my-asg"
            }
          ]
        },
        "Period": 3600,
        "Stat": "Sum"
      },
      "ReturnData": false
    },
    {
      "Id": "m2",
      "Label": "ForecastedLoadValues",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/AutoScaling",
          "MetricName": "PredictiveScalingLoadForecast",
          "Dimensions": [
            {
              "Name": "AutoScalingGroupName",
              "Value": "my-asg"
            },
```

```
            {
              "Name": "PolicyName",
              "Value": "my-predictive-scaling-policy"
            },
            {
              "Name": "PairIndex",
              "Value": "0"
            }
          ]
        },
        "Period": 3600,
        "Stat": "Average"
      },
      "ReturnData": false
    }
  ]
}
```

Instead of a single metric, there is an array of metric data query structures for `MetricDataQueries`. Each item in `MetricDataQueries` gets a metric or performs a math expression. The first item, `e1`, is the math expression. The designated expression sets the `ReturnData` parameter to `true`, which ultimately produces a single time series. For all other metrics, the `ReturnData` value is `false`.

In the example, the designated expression uses the actual and forecasted values as input and returns the new metric (MAPE). `m1` is the CloudWatch metric that contains the actual load values (assuming CPU utilization is the load metric that was originally specified for the policy named `my-predictive-scaling-policy`). `m2` is the CloudWatch metric that contains the forecasted load values. The math syntax for the MAPE metric is as follows:

Average of (abs ((Actual - Forecast)/(Actual)))

Visualize your accuracy metrics and set alarms

To visualize the accuracy metric data, select the **Metrics** tab in the CloudWatch console. You can graph the data from there. For more information, see Adding a math expression to a CloudWatch graph in the *Amazon CloudWatch User Guide*.

You can also set an alarm on a metric that you're monitoring from the **Metrics** section. While on the **Graphed metrics** tab, select the **Create alarm** icon under the **Actions** column. The **Create alarm** icon is represented as a small bell. For more information and notification options, see

Creating a CloudWatch alarm based on a metric math expression and Notifying users on alarm changes in the *Amazon CloudWatch User Guide*.

Alternatively, you can use GetMetricData and PutMetricAlarm to perform calculations using metric math and create alarms based on the output.

Override forecast values using scheduled actions

Sometimes, you might have additional information about your future application requirements that the forecast calculation is unable to take into account. For example, forecast calculations might underestimate the capacity needed for an upcoming marketing event. You can use scheduled actions to temporarily override the forecast during future time periods. The scheduled actions can run on a recurring basis, or at a specific date and time when there are one-time demand fluctuations.

For example, you can create a scheduled action with a higher minimum capacity than what is forecasted. At runtime, Amazon EC2 Auto Scaling updates the minimum capacity of your Auto Scaling group. Because predictive scaling optimizes for capacity, a scheduled action with a minimum capacity that is higher than the forecast values is honored. This prevents capacity from being less than expected. To stop overriding the forecast, use a second scheduled action to return the minimum capacity to its original setting.

The following procedure outlines the steps for overriding the forecast during future time periods.

Topics

- Step 1: (Optional) Analyze time series data
- Step 2: Create two scheduled actions

> **Important**
>
> This topic assumes that you are trying to override the forecast to scale to a higher capacity than what is forecasted. If you need to temporarily decrease capacity without interference from a predictive scaling policy, use *forecast only* mode instead. While in forecast only mode, predictive scaling will continue to generate forecasts, but it will not automatically increase capacity. You can then monitor resource utilization and manually decrease the size of your group as needed. For more information about scaling manually, see Manual scaling for Amazon EC2 Auto Scaling.

Step 1: (Optional) Analyze time series data

Start by analyzing the forecast time series data. This is an optional step, but it is helpful if you want to understand the details of the forecast.

1. **Retrieve the forecast**

   After the forecast is created, you can query for a specific time period in the forecast. The goal of the query is to get a complete view of the time series data for a specific time period.

   Your query can include up to two days of future forecast data. If you have been using predictive scaling for a while, you can also access your past forecast data. However, the maximum time duration between the start and end time is 30 days.

   To get the forecast using the get-predictive-scaling-forecast AWS CLI command, provide the following parameters in the command:

   - Enter the name of the Auto Scaling group in the `--auto-scaling-group-name` parameter.
   - Enter the name of the policy in the `--policy-name` parameter.
   - Enter the start time in the `--start-time` parameter to return only forecast data for after or at the specified time.
   - Enter the end time in the `--end-time` parameter to return only forecast data for before the specified time.

```
aws autoscaling get-predictive-scaling-forecast --auto-scaling-group-name my-asg \
    --policy-name cpu40-predictive-scaling-policy \
    --start-time "2021-05-19T17:00:00Z" \
    --end-time "2021-05-19T23:00:00Z"
```

   If successful, the command returns data similar to the following example.

```
{
    "LoadForecast": [
        {
            "Timestamps": [
                "2021-05-19T17:00:00+00:00",
                "2021-05-19T18:00:00+00:00",
                "2021-05-19T19:00:00+00:00",
```

```
                "2021-05-19T20:00:00+00:00",
                "2021-05-19T21:00:00+00:00",
                "2021-05-19T22:00:00+00:00",
                "2021-05-19T23:00:00+00:00"
            ],
            "Values": [
                153.0655799339254,
                128.8288551285919,
                107.1179447150675,
                197.3601844551528,
                626.4039934516954,
                596.9441277518481,
                677.9675713779869
            ],
            "MetricSpecification": {
                "TargetValue": 40.0,
                "PredefinedMetricPairSpecification": {
                    "PredefinedMetricType": "ASGCPUUtilization"
                }
            }
        }
    ],
    "CapacityForecast": {
        "Timestamps": [
            "2021-05-19T17:00:00+00:00",
            "2021-05-19T18:00:00+00:00",
            "2021-05-19T19:00:00+00:00",
            "2021-05-19T20:00:00+00:00",
            "2021-05-19T21:00:00+00:00",
            "2021-05-19T22:00:00+00:00",
            "2021-05-19T23:00:00+00:00"
        ],
        "Values": [
            2.0,
            2.0,
            2.0,
            2.0,
            4.0,
            4.0,
            4.0
        ]
    },
    "UpdateTime": "2021-05-19T01:52:50.118000+00:00"
```

```
}
```

The response includes two forecasts: `LoadForecast` and `CapacityForecast`. `LoadForecast` shows the hourly load forecast. `CapacityForecast` shows forecast values for the capacity that is needed on an hourly basis to handle the forecasted load while maintaining a `TargetValue` of 40.0 (40% average CPU utilization).

2. **Identify the target time period**

   Identify the hour or hours when the one-time demand fluctuation should take place. Remember that dates and times shown in the forecast are in UTC.

Step 2: Create two scheduled actions

Next, create two scheduled actions for a specific time period when your application will have a higher than forecasted load. For example, if you have a marketing event that will drive traffic to your site for a limited period of time, you can schedule a one-time action to update the minimum capacity when it starts. Then, schedule another action to return the minimum capacity to the original setting when the event ends.

To create two scheduled actions for one-time events (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Automatic scaling** tab, in **Scheduled actions**, choose **Create scheduled action**.
4. Fill in the following scheduled action settings:

   a. Enter a **Name** for the scheduled action.

   b. For **Min**, enter the new minimum capacity for your Auto Scaling group. The **Min** must be less than or equal to the maximum size of the group. If your value for **Min** is greater than group's maximum size, you must update **Max**.

   c. For **Recurrence**, choose **Once**.

   d. For **Time zone**, choose a time zone. If no time zone is chosen, ETC/UTC is used by default.

   e. Define a **Specific start time**.

5. Choose **Create**.

   The console displays the scheduled actions for the Auto Scaling group.

6. Configure a second scheduled action to return the minimum capacity to the original setting at the end of the event. Predictive scaling can scale capacity only when the value you set for **Min** is lower than the forecast values.

To create two scheduled actions for one-time events (AWS CLI)

To use the AWS CLI to create the scheduled actions, use the put-scheduled-update-group-action command.

For example, let's define a schedule that maintains a minimum capacity of three instances on May 19 at 5:00 PM for eight hours. The following commands show how to implement this scenario.

The first put-scheduled-update-group-action command instructs Amazon EC2 Auto Scaling to update the minimum capacity of the specified Auto Scaling group at 5:00 PM UTC on May 19, 2021.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-start \
  --auto-scaling-group-name my-asg --start-time "2021-05-19T17:00:00Z" --minimum-capacity 3
```

The second command instructs Amazon EC2 Auto Scaling to set the group's minimum capacity to one at 1:00 AM UTC on May 20, 2021.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-end \
  --auto-scaling-group-name my-asg --start-time "2021-05-20T01:00:00Z" --minimum-capacity 1
```

After you add these scheduled actions to the Auto Scaling group, Amazon EC2 Auto Scaling does the following:

- At 5:00 PM UTC on May 19, 2021, the first scheduled action runs. If the group currently has fewer than three instances, the group scales out to three instances. During this time and for the next eight hours, Amazon EC2 Auto Scaling can continue to scale out if the predicted capacity is higher than the actual capacity or if there is a dynamic scaling policy in effect.

- At 1:00 AM UTC on May 20, 2021, the second scheduled action runs. This returns the minimum capacity to its original setting at the end of the event.

Scaling based on recurring schedules

To override the forecast for the same time period every week, create two scheduled actions and provide the time and date logic using a cron expression.

The cron expression format consists of five fields separated by spaces: [Minute] [Hour] [Day_of_Month] [Month_of_Year] [Day_of_Week]. Fields can contain any allowed values, including special characters.

For example, the following cron expression runs the action every Tuesday at 6:30 AM. The asterisk is used as a wildcard to match all values for a field.

```
30 6 * * 2
```

See also

For more information about how to create, list, edit, and delete scheduled actions, see Scheduled scaling for Amazon EC2 Auto Scaling.

Advanced predictive scaling policy using custom metrics

In a predictive scaling policy, you can use predefined or custom metrics. Custom metrics are useful when the predefined metrics (CPU, network I/O, and Application Load Balancer request count) do not sufficiently describe your application load.

When creating a predictive scaling policy with custom metrics, you can specify other CloudWatch metrics provided by AWS, or you can specify metrics that you define and publish yourself. You can also use metric math to aggregate and transform existing metrics into a new time series that AWS doesn't automatically track. When you combine values in your data, for example, by calculating new sums or averages, it's called *aggregating*. The resulting data is called an *aggregate*.

The following section contains best practices and examples of how to construct the JSON structure for the policy.

Topics

- Best practices
- Prerequisites

- Constructing the JSON for custom metrics
- Considerations for custom metrics in a predictive scaling policy
- Limitations

Best practices

The following best practices can help you use custom metrics more effectively:

- For the load metric specification, the most useful metric is a metric that represents the load on an Auto Scaling group as a whole, regardless of the group's capacity.
- For the scaling metric specification, the most useful metric to scale by is an average throughput or utilization per instance metric.
- The scaling metric must be inversely proportional to capacity. That is, if the number of instances in the Auto Scaling group increases, the scaling metric should decrease by roughly the same proportion. To ensure that predictive scaling behaves as expected, the load metric and the scaling metric must also correlate strongly with each other.
- The target utilization must match the type of scaling metric. For a policy configuration that uses CPU utilization, this is a target percentage. For a policy configuration that uses throughput, such as the number of requests or messages, this is the target number of requests or messages per instance during any one-minute interval.
- If these recommendations are not followed, the forecasted future values of the time series will probably be incorrect. To validate that the data is correct, you can view the forecasted values in the Amazon EC2 Auto Scaling console. Alternatively, after you create your predictive scaling policy, inspect the `LoadForecast` and `CapacityForecast` objects returned by a call to the GetPredictiveScalingForecast API.
- We strongly recommend that you configure predictive scaling in forecast only mode so that you can evaluate the forecast before predictive scaling starts actively scaling capacity.

Prerequisites

To add custom metrics to your predictive scaling policy, you must have `cloudwatch:GetMetricData` permissions.

To specify your own metrics instead of the metrics that AWS provides, you must first publish your metrics to CloudWatch. For more information, see Publishing custom metrics in the *Amazon CloudWatch User Guide*.

If you publish your own metrics, make sure to publish the data points at a minimum frequency of five minutes. Amazon EC2 Auto Scaling retrieves the data points from CloudWatch based on the length of the period that it needs. For example, the load metric specification uses hourly metrics to measure the load on your application. CloudWatch uses your published metric data to provide a single data value for any one-hour period by aggregating all data points with timestamps that fall within each one-hour period.

Constructing the JSON for custom metrics

The following section contains examples for how to configure predictive scaling to query data from CloudWatch. There are two different methods to configure this option, and the method that you choose affects which format you use to construct the JSON for your predictive scaling policy. When you use metric math, the format of the JSON varies further based on the metric math being performed.

1. To create a policy that gets data directly from other CloudWatch metrics provided by AWS or metrics that you publish to CloudWatch, see Example predictive scaling policy with custom load and scaling metrics (AWS CLI).
2. To create a policy that can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics, see Use metric math expressions.

Example predictive scaling policy with custom load and scaling metrics (AWS CLI)

To create a predictive scaling policy with custom load and scaling metrics with the AWS CLI, store the arguments for `--predictive-scaling-configuration` in a JSON file named `config.json`.

You start adding custom metrics by replacing the replaceable values in the following example with those of your metrics and your target utilization.

```
{
  "MetricSpecifications": [
    {
      "TargetValue": 50,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "scaling_metric",
            "MetricStat": {
```

```
                "Metric": {
                  "MetricName": "MyUtilizationMetric",
                  "Namespace": "MyNameSpace",
                  "Dimensions": [
                    {
                      "Name": "MyOptionalMetricDimensionName",
                      "Value": "MyOptionalMetricDimensionValue"
                    }
                  ]
                },
                "Stat": "Average"
              }
            }
          ]
        },
        "CustomizedLoadMetricSpecification": {
          "MetricDataQueries": [
            {
              "Id": "load_metric",
              "MetricStat": {
                "Metric": {
                  "MetricName": "MyLoadMetric",
                  "Namespace": "MyNameSpace",
                  "Dimensions": [
                    {
                      "Name": "MyOptionalMetricDimensionName",
                      "Value": "MyOptionalMetricDimensionValue"
                    }
                  ]
                },
                "Stat": "Sum"
              }
            }
          ]
        }
      }
    ]
}
```

For more information, see MetricDataQuery in the *Amazon EC2 Auto Scaling API Reference*.

> **Note**
>
> Following are some additional resources that can help you find metric names, namespaces, dimensions, and statistics for CloudWatch metrics:
>
> - For information about the available metrics for AWS services, see AWS services that publish CloudWatch metrics in the *Amazon CloudWatch User Guide*.
> - To get the exact metric name, namespace, and dimensions (if applicable) for a CloudWatch metric with the AWS CLI, see list-metrics.

To create this policy, run the put-scaling-policy command using the JSON file as input, as demonstrated in the following example.

```
aws autoscaling put-scaling-policy --policy-name my-predictive-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN).

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-predictive-scaling-policy",
  "Alarms": []
}
```

Use metric math expressions

The following section provides information and examples of predictive scaling policies that show how you can use metric math in your policy.

Topics

- Understand metric math
- Example predictive scaling policy that combines metrics using metric math (AWS CLI)
- Example predictive scaling policy to use in a blue/green deployment scenario (AWS CLI)

Understand metric math

If all you want to do is aggregate existing metric data, CloudWatch metric math saves you the effort and cost of publishing another metric to CloudWatch. You can use any metric that AWS provides, and you can also use metrics that you define as part of your applications. For example, you might want to calculate the Amazon SQS queue backlog per instance. You can do this by taking the approximate number of messages available for retrieval from the queue and dividing that number by the Auto Scaling group's running capacity.

For more information, see Using metric math in the *Amazon CloudWatch User Guide*.

If you choose to use a metric math expression in your predictive scaling policy, consider the following points:

- Metric math operations use the data points of the unique combination of metric name, namespace, and dimension keys/value pairs of metrics.
- You can use any arithmetic operator (+ - * / ^), statistical function (such as AVG or SUM), or other function that CloudWatch supports.
- You can use both metrics and the results of other math expressions in the formulas of the math expression.
- Your metric math expressions can be made up of different aggregations. However, it's a best practice for the final aggregation result to use `Average` for the scaling metric and `Sum` for the load metric.
- Any expressions used in a metric specification must eventually return a single time series.

To use metric math, do the following:

- Choose one or more CloudWatch metrics. Then, create the expression. For more information, see Using metric math in the *Amazon CloudWatch User Guide*.
- Verify that the metric math expression is valid by using the CloudWatch console or the CloudWatch GetMetricData API.

Example predictive scaling policy that combines metrics using metric math (AWS CLI)

Sometimes, instead of specifying the metric directly, you might need to first process its data in some way. For example, you might have an application that pulls work from an Amazon SQS queue, and you might want to use the number of items in the queue as criteria for predictive

scaling. The number of messages in the queue does not solely define the number of instances that you need. Therefore, more work is needed to create a metric that can be used to calculate the backlog per instance. For more information, see Scaling policy based on Amazon SQS.

The following is an example predictive scaling policy for this scenario. It specifies scaling and load metrics that are based on the Amazon SQS `ApproximateNumberOfMessagesVisible` metric, which is the number of messages available for retrieval from the queue. It also uses the Amazon EC2 Auto Scaling `GroupInServiceInstances` metric and a math expression to calculate the backlog per instance for the scaling metric.

```
aws autoscaling put-scaling-policy --policy-name my-sqs-custom-metrics-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
{
  "MetricSpecifications": [
    {
      "TargetValue": 100,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Label": "Get the queue size (the number of messages waiting to be
 processed)",
            "Id": "queue_size",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Get the group size (the number of running instances)",
            "Id": "running_capacity",
            "MetricStat": {
              "Metric": {
```

```
                "MetricName": "GroupInServiceInstances",
                "Namespace": "AWS/AutoScaling",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "my-asg"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Calculate the backlog per instance",
            "Id": "scaling_metric",
            "Expression": "queue_size / running_capacity",
            "ReturnData": true
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ],
              },
              "Stat": "Sum"
            },
            "ReturnData": true
          }
        ]
      }
    }
  ]
```

```
}
```

The example returns the policy's ARN.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-sqs-custom-metrics-policy",
  "Alarms": []
}
```

Example predictive scaling policy to use in a blue/green deployment scenario (AWS CLI)

A search expression provides an advanced option in which you can query for a metric from multiple Auto Scaling groups and perform math expressions on them. This is especially useful for blue/green deployments.

> **Note**
>
> A *blue/green deployment* is a deployment method in which you create two separate but identical Auto Scaling groups. Only one of the groups receives production traffic. User traffic is initially directed to the earlier ("blue") Auto Scaling group, while a new group ("green") is used for testing and evaluation of a new version of an application or service. User traffic is shifted to the green Auto Scaling group after a new deployment is tested and accepted. You can then delete the blue group after the deployment is successful.

When new Auto Scaling groups get created as part of a blue/green deployment, the metric history of each group can be automatically included in the predictive scaling policy without you having to change its metric specifications. For more information, see Using EC2 Auto Scaling predictive scaling policies with Blue/Green deployments on the AWS Compute Blog.

The following example policy shows how this can be done. In this example, the policy uses the `CPUUtilization` metric emitted by Amazon EC2. It uses the Amazon EC2 Auto Scaling `GroupInServiceInstances` metric and a math expression to calculate the value of the scaling metric per instance. It also specifies a capacity metric specification to get the `GroupInServiceInstances` metric.

The search expression finds the `CPUUtilization` of instances in multiple Auto Scaling groups based on the specified search criteria. If you later create a new Auto Scaling group that matches

the same search criteria, the `CPUUtilization` of the instances in the new Auto Scaling group is automatically included.

```
aws autoscaling put-scaling-policy --policy-name my-blue-green-predictive-scaling-
policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
{
  "MetricSpecifications": [
    {
      "TargetValue": 25,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_sum",
            "Expression": "SUM(SEARCH('{AWS/EC2,AutoScalingGroupName} MetricName=
\"CPUUtilization\" ASG-myapp', 'Sum', 300))",
            "ReturnData": false
          },
          {
            "Id": "capacity_sum",
            "Expression": "SUM(SEARCH('{AWS/AutoScaling,AutoScalingGroupName}
 MetricName=\"GroupInServiceInstances\" ASG-myapp', 'Average', 300))",
            "ReturnData": false
          },
          {
            "Id": "weighted_average",
            "Expression": "load_sum / capacity_sum",
            "ReturnData": true
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_sum",
            "Expression": "SUM(SEARCH('{AWS/EC2,AutoScalingGroupName} MetricName=
\"CPUUtilization\" ASG-myapp', 'Sum', 3600))"
          }
        ]
      },
      "CustomizedCapacityMetricSpecification": {
        "MetricDataQueries": [
```

```
          {
            "Id": "capacity_sum",
            "Expression": "SUM(SEARCH('{AWS/AutoScaling,AutoScalingGroupName}
 MetricName=\"GroupInServiceInstances\" ASG-myapp', 'Average', 300))"
          }
        ]
      }
    }
  ]
}
```

The example returns the policy's ARN.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-blue-green-predictive-
scaling-policy",
  "Alarms": []
}
```

Considerations for custom metrics in a predictive scaling policy

If an issue occurs while using custom metrics, we recommend that you do the following:

- If an error message is provided, read the message and resolve the issue it reports, if possible.
- If an issue occurs when you are trying to use a search expression in a blue/green deployment scenario, first make sure that you understand how to create a search expression that looks for a partial match instead of an exact match. Also, check that your query finds only the Auto Scaling groups that are running the specific application. For more information about the search expression syntax, see CloudWatch search expression syntax in the *Amazon CloudWatch User Guide*.
- If you did not validate an expression in advance, the put-scaling-policy command validates it when you create your scaling policy. However, there is a possibility that this command might fail to identify the exact cause of the detected errors. To fix the issues, troubleshoot the errors that you receive in a response from a request to the get-metric-data command. You can also troubleshoot the expression from the CloudWatch console.
- When you view your **Load** and **Capacity** graphs in the console, the **Capacity** graph might not show any data. To ensure that the graphs have complete data, make sure that you consistently

enable group metrics for your Auto Scaling groups. For more information, see Enable Auto Scaling group metrics (console).

- The capacity metric specification is only useful for blue/green deployments when you have applications that run in different Auto Scaling groups over their lifetime. This custom metric lets you provide the total capacity of multiple Auto Scaling groups. Predictive scaling uses this to show historical data in the **Capacity** graphs in the console.
- You must specify `false` for `ReturnData` if `MetricDataQueries` specifies the SEARCH() function on its own without a math function like SUM(). This is because search expressions might return multiple time series, and a metric specification based on an expression can return only one time series.
- All metrics involved in a search expression should be of the same resolution.

Limitations

- You can query data points of up to 10 metrics in one metric specification.
- For the purposes of this limit, one expression counts as one metric.

Control which Auto Scaling instances terminate during scale in

Amazon EC2 Auto Scaling uses termination policies to decide the order for terminating instances. You can use a predefined policy or create a custom policy to meet your specific requirements. By using a custom policy or instance scale in protection, you can also prevent your Auto Scaling group from terminating instances that aren't yet ready to terminate.

Contents

- When Amazon EC2 Auto Scaling uses termination policies
- Configure termination policies for Amazon EC2 Auto Scaling
- Create a custom termination policy with Lambda
- Use instance scale-in protection to control instance termination
- Design your applications to gracefully handle instance termination

When Amazon EC2 Auto Scaling uses termination policies

The following sections describe the scenarios in which Amazon EC2 Auto Scaling uses termination policies.

Contents

- Scale in events
- Instance refresh
- Availability Zone rebalancing

Scale in events

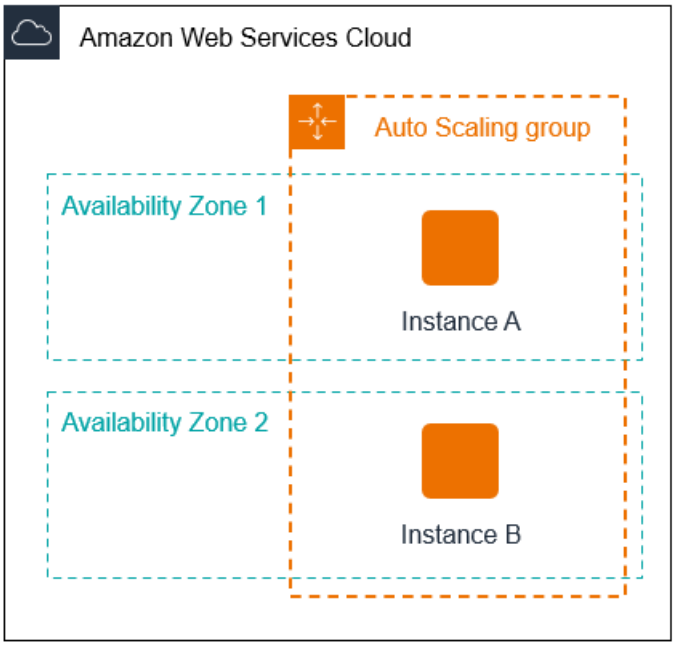
A scale in event occurs when there is a new value for the desired capacity of an Auto Scaling group that is lower than the current capacity of the group.

scale in events occur in the following scenarios:

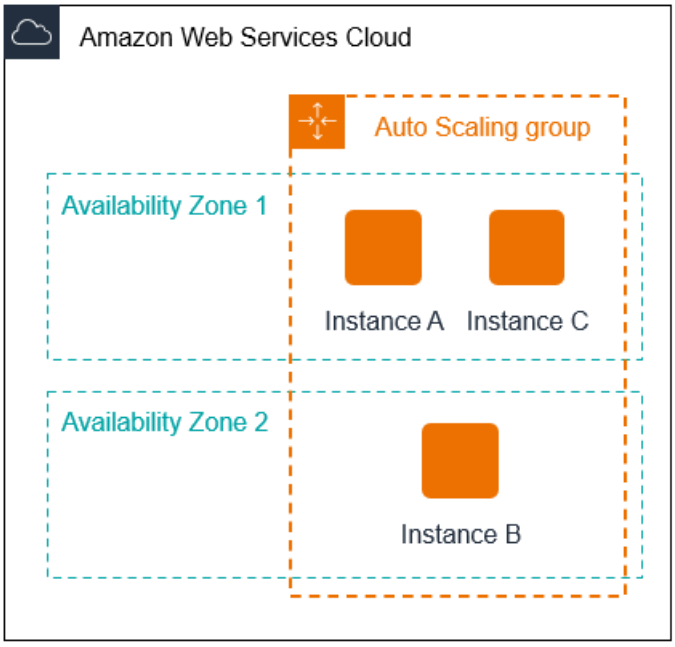
- When using dynamic scaling policies and the size of the group decreases as a result of changes in a metric's value
- When using scheduled scaling and the size of the group decreases as a result of a scheduled action
- When you manually decrease the size of the group

The following example shows how termination policies work when there is a scale in event.

1. The Auto Scaling group in this example has one instance type, two Availability Zones, and a desired capacity of two instances. It also has a dynamic scaling policy that adds and removes instances when resource utilization increases or decreases. The two instances in this group are distributed across the two Availability Zones as shown in the following diagram.

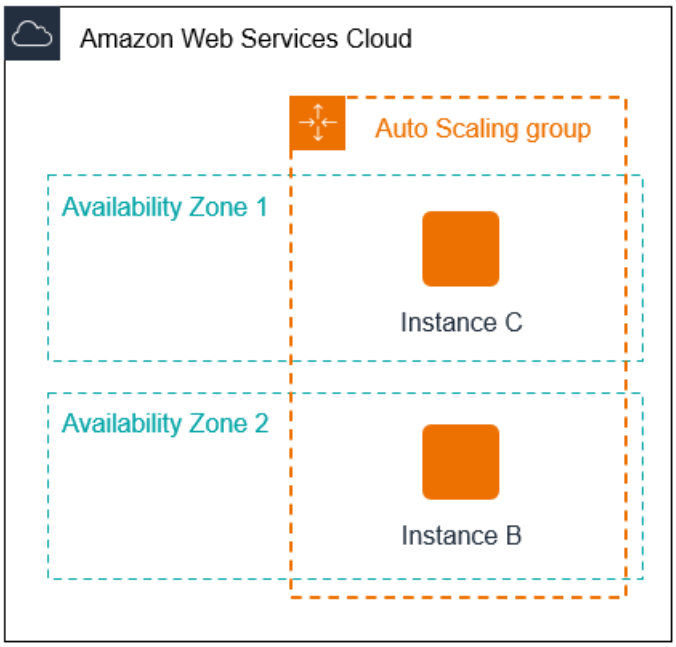


2. When the Auto Scaling group scales out, Amazon EC2 Auto Scaling launches a new instance. The Auto Scaling group now has three instances, distributed across the two Availability Zones as shown in the following diagram.



3. When the Auto Scaling group scales in, Amazon EC2 Auto Scaling terminates one of the instances.
4. If you did not assign a specific termination policy to the group, Amazon EC2 Auto Scaling uses the default termination policy. It selects the Availability Zone with two instances, and terminates the instance that was launched from a launch configuration, a different launch template, or the

oldest version of the current launch template. If the instances were launched from the same launch template and version, Amazon EC2 Auto Scaling selects the instance that is closest to the next billing hour and terminates it.



Instance refresh

You can start an instance refresh to update the instances in your Auto Scaling group. During an instance refresh, Amazon EC2 Auto Scaling terminates instances in the group and then launches replacements for the terminated instances. The termination policy for the Auto Scaling group controls which instances are replaced first.

Availability Zone rebalancing

Amazon EC2 Auto Scaling balances your capacity evenly across the Availability Zones enabled for your Auto Scaling group. This helps reduce the impact of an Availability Zone outage. If the distribution of capacity across Availability Zones becomes out of balance, Amazon EC2 Auto Scaling rebalances the Auto Scaling group by launching instances in the enabled Availability Zones with the fewest instances and terminating instances elsewhere. The termination policy controls which instances are prioritized for termination first.

There are a number of reasons why the distribution of instances across Availability Zones can become out of balance.

Removing instances

If you detach instances from your Auto Scaling group, you put instances on standby, or you explicitly terminate instances and decrement the desired capacity, which prevents replacement instances from launching, the group can become unbalanced. If this occurs, Amazon EC2 Auto Scaling compensates by rebalancing the Availability Zones.

Using different Availability Zones than originally specified

If you expand your Auto Scaling group to include additional Availability Zones, or you change which Availability Zones are used, Amazon EC2 Auto Scaling launches instances in the new Availability Zones and terminates instances in other zones to help ensure that your Auto Scaling group spans Availability Zones evenly.

Availability outage

Availability outages are rare. However, if one Availability Zone becomes unavailable and recovers later, your Auto Scaling group can become unbalanced between Availability Zones. Amazon EC2 Auto Scaling tries to gradually rebalance the group, and rebalancing might terminate instances in other zones.

For example, imagine that you have an Auto Scaling group that has one instance type, two Availability Zones, and a desired capacity of two instances. In a situation where one Availability Zone fails, Amazon EC2 Auto Scaling automatically launches a new instance in the healthy Availability Zone to replace the one in the unhealthy Availability Zone. Then, when the unhealthy Availability Zone returns to a healthy state later on, Amazon EC2 Auto Scaling automatically launches a new instance in this zone, which in turn terminates an instance in the unaffected zone.

Note

When rebalancing, Amazon EC2 Auto Scaling launches new instances before terminating the old ones, so that rebalancing does not compromise the performance or availability of your application.
Because Amazon EC2 Auto Scaling attempts to launch new instances before terminating the old ones, being at or near the specified maximum capacity could impede or completely stop rebalancing activities. To avoid this problem, the system can temporarily exceed the specified maximum capacity of a group by a 10 percent margin (or by a margin of one instance, whichever is greater) during a rebalancing activity. The margin is extended only if

the group is at or near maximum capacity and needs rebalancing, either because of user-requested rezoning or to compensate for zone availability issues. The extension lasts only as long as needed to rebalance the group.

Configure termination policies for Amazon EC2 Auto Scaling

A termination policy provides the criteria that Amazon EC2 Auto Scaling follows to terminate instances in a specific order. By default, Amazon EC2 Auto Scaling uses a termination policy that's designed to terminate instances that are using outdated configurations first. You can change the termination policy to control which instances are most important to terminate first.

When Amazon EC2 Auto Scaling terminates instances, it tries to maintain balance across the Availability Zones that are enabled for your Auto Scaling group. Maintaining Zonal balance takes precedence over the termination policy. If one Availability Zone has more instances than others, Amazon EC2 Auto Scaling applies the termination policy to the imbalanced zone first. If the Availability Zones are balanced, it applies the termination policy across all Zones.

Topics

- How the default termination policy works
- Default termination policy and mixed instances groups
- Predefined termination policies
- Change the termination policy for an Auto Scaling group

How the default termination policy works

When Amazon EC2 Auto Scaling needs to terminate an instance, it first identifies which Availability Zone (or Zones) has the most instances and at least one instance that is not protected from scale in. Then, it proceeds to evaluate unprotected instances within the identified Availability Zone as follows:

Instances that use outdated configurations

- **For groups that use a launch template** – Determine whether any of the instances use outdated configurations, prioritizing in this order:
  1. First, check for instances launched with a launch configuration.

2. Then, check for instances launched using a different launch template instead of the current launch template.
3. Finally, check for instances using the oldest version of the current launch template.

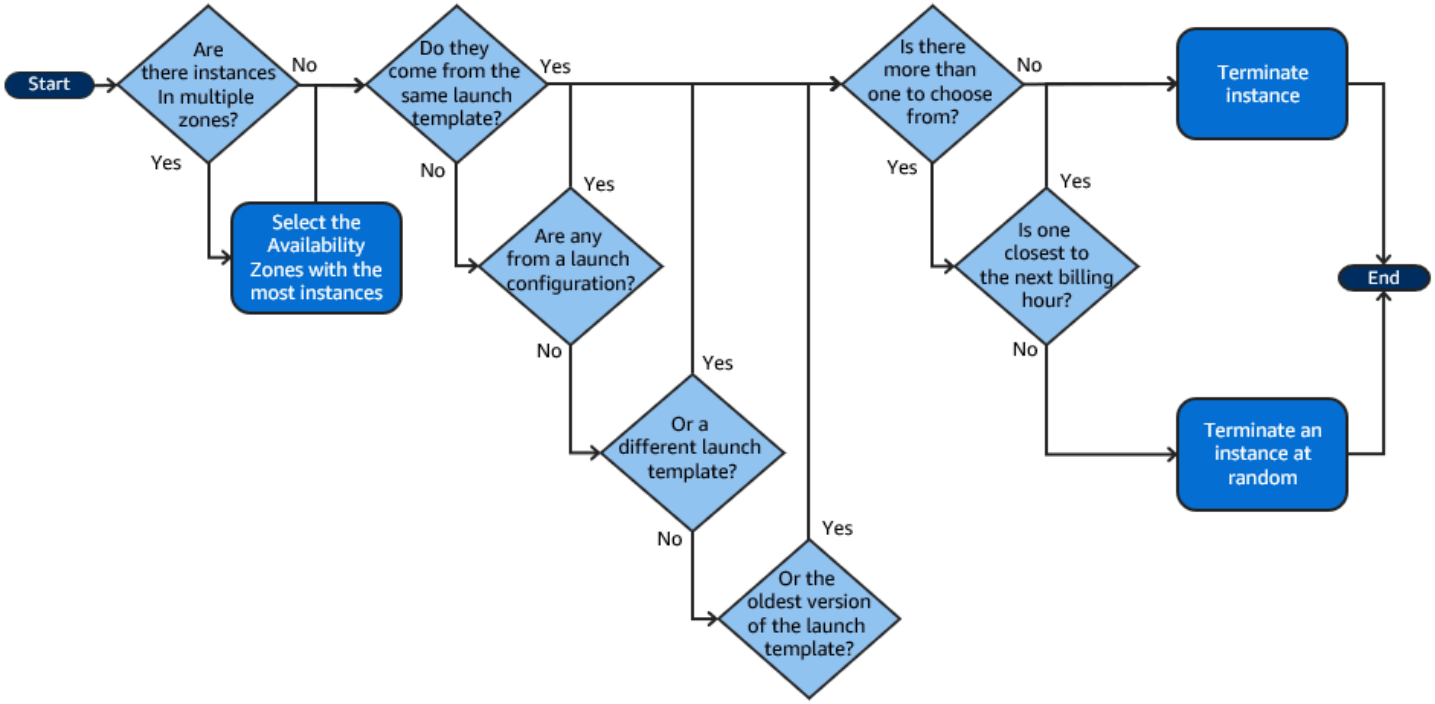
- **For groups that use a launch configuration** – Determine whether any of the instances use the oldest launch configuration.

If no instances with outdated configurations are found, or there are multiple instances to choose from, Amazon EC2 Auto Scaling considers the next criteria of instances approaching their next billing hour.

Instances approaching next billing hour

Determine whether any of the instances that meet the previous criteria are closest to the next billing hour. If multiple instances are equally close, terminate one at random. This helps you maximize the use of your instances that are billed hourly. However, most EC2 usage is now billed per second, so this optimization provides less benefit. For more information, see Amazon EC2 pricing.

The following flow diagram illustrates how the default termination policy works for groups that use a launch template.



Default termination policy and mixed instances groups

Amazon EC2 Auto Scaling applies additional criteria when terminating instances in mixed instances groups.

When Amazon EC2 Auto Scaling needs to terminate an instance, it first identifies which purchase option (Spot or On-Demand) should be terminated based on the settings of the group. This makes sure that the group trends toward the specified ratio of Spot and On-Demand instances over time.

It then applies the termination policy independently within each Availability Zone. It determines which Spot or On-Demand Instance in which Availability Zone to terminate to keep the Availability Zones balanced. The same logic applies to a mixed instances group with weights defined for the instance types.

Within each zone, the default termination policy works as follows to determine which unprotected instance within the identified purchase option can be terminated:

1. Determine whether any of the instances can be terminated to improve alignment with the specified allocation strategy for the Auto Scaling group. If no instances are identified for optimization, or there are multiple instances to choose from, the evaluation continues.
2. Determine whether any of the instances use outdated configurations, prioritizing in this order:
   a. First, check for instances launched with a launch configuration.
   b. Then, check for instances launched using a different launch template instead of the current launch template.
   c. Finally, check for instances using the oldest version of the current launch template.

   If no instances with outdated configurations are found, or there are multiple instances to choose from, the evaluation continues.
3. Determine whether any of the instances are closest to the next billing hour. If multiple instances are equally close, choose one at random.

Predefined termination policies

You choose from the following predefined termination policies:

- **`Default`** – Terminate instances according to the default termination policy.
- **`AllocationStrategy`** – Terminate instances in the Auto Scaling group to align the remaining instances to the allocation strategy for the type of instance that is terminating (either a Spot

Instance or an On-Demand Instance). This policy is useful when your preferred instance types have changed. If the Spot allocation strategy is `lowest-price`, you can gradually rebalance the distribution of Spot Instances across your N lowest priced Spot pools. If the Spot allocation strategy is `capacity-optimized`, you can gradually rebalance the distribution of Spot Instances across Spot pools where there is more available Spot capacity. You can also gradually replace On-Demand Instances of a lower priority type with On-Demand Instances of a higher priority type.

- **`OldestLaunchTemplate`** – Terminate instances that have the oldest launch template. With this policy, instances that use the noncurrent launch template are terminated first, followed by instances that use the oldest version of the current launch template. This policy is useful when you're updating a group and phasing out the instances from a previous configuration.
- **`OldestLaunchConfiguration`** – Terminate instances that have the oldest launch configuration. This policy is useful when you're updating a group and phasing out the instances from a previous configuration. With this policy, instances that use the noncurrent launch configuration are terminated first.
- **`ClosestToNextInstanceHour`** – Terminate instances that are closest to the next billing hour. This policy helps you maximize the use of your instances that have an hourly charge.
- **`NewestInstance`** – Terminate the newest instance in the group. This policy is useful when you're testing a new launch configuration but don't want to keep it in production.
- **`OldestInstance`** – Terminate the oldest instance in the group. This option is useful when you're upgrading the instances in the Auto Scaling group to a new EC2 instance type. You can gradually replace instances of the old type with instances of the new type.

> **Note**
>
> Amazon EC2 Auto Scaling always balances instances across Availability Zones first, regardless of which termination policy is used. As a result, you might encounter situations in which some newer instances are terminated before older instances. For example, when there is a more recently added Availability Zone, or when one Availability Zone has more instances than the other Availability Zones that are used by the group.

Change the termination policy for an Auto Scaling group

To change the termination policy for your Auto Scaling group, use one of the following methods.

Console

You can't change the termination policy when you initially create an Auto Scaling group in the Amazon EC2 Auto Scaling console. The default termination policy is used automatically. After your Auto Scaling group has been created, you can replace the default policy with a different termination policy or multiple termination policies listed in the order in which they should apply.

To change the termination policy for an Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Details** tab, choose **Advanced configurations**, **Edit**.
4. For **Termination policies**, choose one or more termination policies. If you choose multiple policies, put them in the order that you want them evaluated in.

   You can optionally choose **Custom termination policy** and then choose a Lambda function that meets your needs. If you have created versions and aliases for your Lambda function, you can choose a version or alias from the **Version/Alias** drop-down. To use the unpublished version of your Lambda function, keep **Version/Alias** set to its default. For more information, see Create a custom termination policy with Lambda.

   > **Note**
   >
   > When using multiple policies, their order must be set correctly:
   >
   > - If you use the **Default** policy, it must be the last policy in the list.
   > - If you use a **Custom termination policy**, it must be the first policy in the list.

5. Choose **Update**.

AWS CLI

The default termination policy is used automatically unless a different policy is specified.

To change the termination policy for an Auto Scaling group

Use one of the following commands:

- create-auto-scaling-group
- update-auto-scaling-group

You can use termination policies individually, or combine them into a list of policies. For example, use the following command to update an Auto Scaling group to use the `OldestLaunchConfiguration` policy first and then use the `ClosestToNextInstanceHour` policy.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg --
termination-policies "OldestLaunchConfiguration" "ClosestToNextInstanceHour"
```

If you use the `Default` termination policy, make it the last one in the list of termination policies. For example, `--termination-policies "OldestLaunchConfiguration" "Default"`.

To use a custom termination policy, you must first create your termination policy using AWS Lambda. To specify the Lambda function to use as your termination policy, make it the first one in the list of termination policies. For example, `--termination-policies "arn:aws:lambda:us-west-2:123456789012:function:HelloFunction:prod" "OldestLaunchConfiguration"`. For more information, see Create a custom termination policy with Lambda.

Create a custom termination policy with Lambda

Amazon EC2 Auto Scaling uses termination policies to prioritize which instances to terminate first when decreasing the size of your Auto Scaling group (referred to as *scaling in*). Your Auto Scaling group uses a default termination policy, but you can optionally choose or create your own termination policies. For more information about choosing a predefined termination policy, see Configure termination policies for Amazon EC2 Auto Scaling.

In this topic, you learn how to create a custom termination policy using an AWS Lambda function that Amazon EC2 Auto Scaling invokes in response to certain events. The Lambda function that you create processes the information in the input data sent by Amazon EC2 Auto Scaling and returns a list of instances that are ready to terminate.

A custom termination policy provides better control over which instances are terminated, and when. For example, when your Auto Scaling group scales in, Amazon EC2 Auto Scaling cannot determine whether there are workloads running that should not be disrupted. With a Lambda function, you can validate the termination request and wait until the workload is done before returning the instance ID to Amazon EC2 Auto Scaling for termination.

Contents

- Input data
- Response data
- Considerations
- Create the Lambda function
- Limitations

Input data

Amazon EC2 Auto Scaling generates a JSON payload for scale in events, and also does so when instances are about to be terminated as a result of the maximum instance lifetime or instance refresh features. It also generates a JSON payload for the scale in events that it can initiate when rebalancing your group across Availability Zones.

This payload contains information about the capacity Amazon EC2 Auto Scaling needs to terminate, a list of instances that it suggests for termination, and the event that initiated the termination.

The following is an example payload:

```
{
  "AutoScalingGroupARN": "arn:aws:autoscaling:us-east-1:<account-
id>:autoScalingGroup:d4738357-2d40-4038-ae7e-b00ae0227003:autoScalingGroupName/my-asg",
  "AutoScalingGroupName": "my-asg",
  "CapacityToTerminate": [
    {
      "AvailabilityZone": "us-east-1b",
      "Capacity": 2,
      "InstanceMarketOption": "on-demand"
    },
    {
      "AvailabilityZone": "us-east-1b",
```

```
      "Capacity": 1,
      "InstanceMarketOption": "spot"
    },
    {
      "AvailabilityZone": "us-east-1c",
      "Capacity": 3,
      "InstanceMarketOption": "on-demand"
    }
  ],
  "Instances": [
    {
      "AvailabilityZone": "us-east-1b",
      "InstanceId": "i-0056faf8da3e1f75d",
      "InstanceType": "t2.nano",
      "InstanceMarketOption": "on-demand"
    },
    {
      "AvailabilityZone": "us-east-1c",
      "InstanceId": "i-02e1c69383a3ed501",
      "InstanceType": "t2.nano",
      "InstanceMarketOption": "on-demand"
    },
    {
      "AvailabilityZone": "us-east-1c",
      "InstanceId": "i-036bc44b6092c01c7",
      "InstanceType": "t2.nano",
      "InstanceMarketOption": "on-demand"
    },
    ...
  ],
  "Cause": "SCALE_IN"
}
```

The payload includes the name of the Auto Scaling group, its Amazon Resource Name (ARN), and the following elements:

- `CapacityToTerminate` describes how much of your Spot or On-Demand capacity is set to be terminated in a given Availability Zone.
- `Instances` represents the instances that Amazon EC2 Auto Scaling suggests for termination based on the information in `CapacityToTerminate`.
- `Cause` describes the event that caused the termination: `SCALE_IN`, `INSTANCE_REFRESH`, `MAX_INSTANCE_LIFETIME`, or `REBALANCE`.

The following information outlines the most significant factors in how Amazon EC2 Auto Scaling generates the `Instances` in the input data:

- Maintaining balance across Availability Zones takes precedence when an instance is terminating due to scale in events and instance refresh-based terminations. Therefore, if one Availability Zone has more instances than the other Availability Zones that are used by the group, the input data contains instances that are eligible for termination only from the imbalanced Availability Zone. If the Availability Zones used by the group are balanced, the input data contains instances from all of the Availability Zones for the group.
- When using a mixed instances policy, maintaining your Spot and On-Demand capacities in balance based on your desired percentages for each purchase option also takes precedence. We first identify which of the two types (Spot or On-Demand) should be terminated. We then identify which instances (within the identified purchase option) in which Availability Zones we can terminate that will result in the Availability Zones being most balanced.

Response data

The input data and response data work together to narrow down the list of instances to terminate.

With the given input, the response from your Lambda function should look like the following example:

```
{
  "InstanceIDs": [
    "i-02e1c69383a3ed501",
    "i-036bc44b6092c01c7",
    ...
  ]
}
```

The `InstanceIDs` in the response represent the instances that are ready to terminate.

Alternatively, you can return a different set of instances that are ready to be terminated, which overrides the instances in the input data. If no instances are ready to terminate when your Lambda function is invoked, you can also choose not to return any instances.

When no instances are ready to terminate, the response from your Lambda function should look like the following example:

```
{
  "InstanceIDs": [ ]
}
```

Considerations

Note the following considerations when using a custom termination policy:

- Returning an instance first in the response data does not guarantee its termination. If more than the required number of instances are returned when your Lambda function is invoked, Amazon EC2 Auto Scaling evaluates each instance against the other termination policies that you specified for your Auto Scaling group. When there are multiple termination policies, it tries to apply the next termination policy in the list, and if there are more instances than are required to terminate, it moves on to the next termination policy, and so on. If no other termination policies are specified, then the default termination policy is used to determine which instances to terminate.
- If no instances are returned or your Lambda function times out, then Amazon EC2 Auto Scaling waits a short time before invoking your function again. For any scale in event, it keeps trying as long as the group's desired capacity is less than its current capacity. For instance refresh-based terminations, it keeps trying for an hour. After that, if it continues to fail to terminate any instances, the instance refresh operation fails. With maximum instance lifetime, Amazon EC2 Auto Scaling keeps trying to terminate the instance that is identified as exceeding its maximum lifetime.
- Because your function is retried repeatedly, make sure to test and fix any permanent errors in your code before using a Lambda function as a custom termination policy.
- If you override the input data with your own list of instances to terminate, and terminating these instances puts the Availability Zones out of balance, Amazon EC2 Auto Scaling gradually rebalances the distribution of capacity across Availability Zones. First, it invokes your Lambda function to see if there are instances that are ready to be terminated so that it can determine whether to start rebalancing. If there are instances ready to be terminated, it launches new instances first. When the instances finish launching, it then detects that your group's current capacity is higher than its desired capacity and initiates a scale in event.
- A custom termination policy does not affect your ability to also use scale in protection to protect certain instances from being terminated. For more information, see Use instance scale-in protection to control instance termination.

Create the Lambda function

Start by creating the Lambda function, so that you can specify its Amazon Resource Name (ARN) in the termination policies for your Auto Scaling group.

To create a Lambda function (console)

1. Open the Functions page on the Lambda console.
2. On the navigation bar at the top of the screen, choose the same Region that you used when you created the Auto Scaling group.
3. Choose **Create function**, **Author from scratch**.
4. Under **Basic information**, for **Function name**, enter the name of your function.
5. Choose **Create function**. You are returned to the function's code and configuration.
6. With your function still open in the console, under **Function code**, paste your code into the editor.
7. Choose **Deploy**.
8. Optionally, create a published version of the Lambda function by choosing the **Versions** tab and then **Publish new version**. To learn more about versioning in Lambda, see Lambda function versions in the *AWS Lambda Developer Guide*.
9. If you chose to publish a version, choose the **Aliases** tab if you want to associate an alias with this version of the Lambda function. To learn more about aliases in Lambda, see Lambda function aliases in the *AWS Lambda Developer Guide*
10. Next, choose the **Configuration** tab and then **Permissions**.
11. Scroll down to **Resource-based policy** and then choose **Add permissions**. A resource-based policy is used to grant permissions to invoke your function to the principal that is specified in the policy. In this case, the principal will be the Amazon EC2 Auto Scaling service-linked role that is associated with the Auto Scaling group.
12. In the **Policy statement** section, configure your permissions:

    a. Choose **AWS account**.

    b. For **Principal**, enter the ARN of the calling service-linked role, for example, **`arn:aws:iam::<aws-account-id>:role/aws-service-role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling`**.

    c. For **Action**, choose **lambda:InvokeFunction**.

    d. For **Statement ID**, enter a unique statement ID, such as **`AllowInvokeByAutoScaling`**.

e. Choose **Save**.

13. After you have followed these instructions, continue on to specify the ARN of your function in the termination policies for your Auto Scaling group as a next step. For more information, see Change the termination policy for an Auto Scaling group.

> **Note**
>
> For examples that you can use as a reference for developing your Lambda function, see the GitHub repository for Amazon EC2 Auto Scaling.

Limitations

- You can only specify one Lambda function in the termination policies for an Auto Scaling group. If there are multiple termination policies specified, the Lambda function must be specified first.
- You can reference your Lambda function using either an unqualified ARN (without a suffix) or a qualified ARN that has either a version or an alias as its suffix. If an unqualified ARN is used (for example, `function:my-function`), your resource-based policy must be created on the unpublished version of your function. If a qualified ARN is used (for example, `function:my-function:1` or `function:my-function:prod`), your resource-based policy must be created on that specific published version of your function.
- You cannot use a qualified ARN with the `$LATEST` suffix. If you try to add a custom termination policy that refers to a qualified ARN with the `$LATEST` suffix, it will result in an error.
- The number of instances provided in the input data is limited to 30,000 instances. If there are more than 30,000 instances that could be terminated, the input data includes `"HasMoreInstances": true` to indicate that the maximum number of instances are returned.
- The maximum run time for your Lambda function is two seconds (2000 milliseconds). As a best practice, you should set the timeout value of your Lambda function based on your expected run time. Lambda functions have a default timeout of three seconds, but this can be decreased.
- If your runtime exceeds the 2-second limit, any scale in action will be on hold until the runtime falls below this threshold. For Lambda functions with consistently longer runtimes, find a way to reduce the runtime, such as by caching the results where they can be retrieved during subsequent Lambda invocations.

Use instance scale-in protection to control instance termination

Instance scale-in protection gives you control over which instances Amazon EC2 Auto Scaling can terminate. A common use case for this feature is scaling container-based workloads. For more information, see Design your applications to gracefully handle instance termination.

By default, instance scale-in protection is disabled when you create an Auto Scaling group. This means that Amazon EC2 Auto Scaling can terminate any instance in the group.

You can protect instances as soon as they launch by enabling the instance scale-in protection setting on your Auto Scaling group. Instance scale-in protection starts when the instance state is `InService`. Then, to control which instances can terminate, disable the scale-in protection setting on individual instances within the Auto Scaling group. By doing so, you can continue to protect certain instances from unwanted terminations.

Topics

- Considerations
- Change scale-in protection for an Auto Scaling group
- Change scale-in protection for an instance

Considerations

The following are considerations when using instance scale-in protection:

- If all instances in an Auto Scaling group are protected from scale in, and a scale in event occurs, its desired capacity is decremented. However, the Auto Scaling group can't terminate the required number of instances until their instance scale in protection settings are disabled. In the AWS Management Console, the **Activity history** for the Auto Scaling group includes the following message if all instances in an Auto Scaling group are protected from scale in when a scale in event occurs: `Could not scale to desired capacity because all remaining instances are protected from scale in.`
- If you detach an instance that is protected from scale in, its instance scale in protection setting is lost. When you attach the instance to the group again, it inherits the current instance scale in protection setting of the group. When Amazon EC2 Auto Scaling launches a new instance or moves an instance from a warm pool into the Auto Scaling group, the instance inherits the instance scale in protection setting of the Auto Scaling group.
- Instance scale-in protection does not protect Auto Scaling instances from the following:

- Health check replacement if the instance fails health checks. For more information, see Health checks for instances in an Auto Scaling group.
- Spot Instance interruptions. A Spot Instance is terminated when capacity is no longer available or the Spot price exceeds your maximum price.
- A Capacity Block reservation ends. Amazon EC2 reclaims the Capacity Block instances even if they are protected from scale in.
- Manual termination through the `terminate-instance-in-auto-scaling-group` command. For more information, see Terminate an instance in your Auto Scaling group (AWS CLI).
- Manual termination through the Amazon EC2 console, CLI commands, and API operations. To protect Auto Scaling instances from manual termination, enable Amazon EC2 termination protection. (This does not prevent Amazon EC2 Auto Scaling from terminating instances or manual termination through the `terminate-instance-in-auto-scaling-group` command.) For information about enabling Amazon EC2 termination protection in a launch template, see Create a launch template using advanced settings.

Change scale-in protection for an Auto Scaling group

You can enable or disable the instance scale-in protection setting for an Auto Scaling group. When you enable it, all new instances launched by the group will have instance scale-in protection enabled.

Enabling or disabling this setting for an Auto Scaling group does not affect existing instances.

Console

To enable scale-in protection for a new Auto Scaling group

When you create the Auto Scaling group, on the **Configure group size and scaling policies** page, under **Instance scale-in protection**, select the **Enable instance scale-in protection** check box.

To enable or disable scale-in protection for an existing group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select check box next to the Auto Scaling group.

A split pane opens up in the bottom of the page.

3. On the **Details** tab, choose **Advanced configurations**, **Edit**.
4. For **Instance scale-in protection**, select or clear the **Enable instance-scale protection** check box to enable or disable this option as required.
5. Choose **Update**.

AWS CLI

To enable scale-in protection for a new Auto Scaling group

Use the following create-auto-scaling-group command to enable instance scale-in protection.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg --new-
instances-protected-from-scale-in ...
```

To enable scale-in protection for an existing group

Use the following update-auto-scaling-group command to enable instance scale-in protection for the specified Auto Scaling group.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg --new-
instances-protected-from-scale-in
```

To disable scale-in protection for an existing group

Use the following command to disable instance scale-in protection for the specified group.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg --no-new-
instances-protected-from-scale-in
```

Change scale-in protection for an instance

By default, an instance gets its instance scale-in protection setting from its Auto Scaling group. However, you can enable or disable instance scale-in protection for individual instances after they launch.

Console

To enable or disable scale-in protection for an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Instance management** tab, in **Instances**, select an instance.
4. To enable instance scale-in protection, choose **Actions**, **Set scale-in protection**. When prompted, choose **Set scale-in protection**.
5. To disable instance scale-in protection, choose **Actions**, **Remove scale-in protection**. When prompted, choose **Remove scale-in protection**.

AWS CLI

To enable scale-in protection for an instance

Use the following set-instance-protection command to enable instance scale-in protection for the specified instance.

```
aws autoscaling set-instance-protection --instance-ids i-5f2e8a0d --auto-scaling-
group-name my-asg --protected-from-scale-in
```

To disable scale-in protection for an instance

Use the following command to disable instance scale-in protection for the specified instance.

```
aws autoscaling set-instance-protection --instance-ids i-5f2e8a0d --auto-scaling-
group-name my-asg --no-protected-from-scale-in
```

> **Note**
>
> Remember, instance scale-in protection does not guarantee that instances won't be terminated in the event of a human error—for example, if someone manually terminates an instance using the Amazon EC2 console or AWS CLI. To protect your instance from accidental termination, you can use Amazon EC2 termination protection. However, even

with termination protection and instance scale-in protection enabled, data saved to instance storage can be lost if a health check determines that an instance is unhealthy or if the group itself is accidentally deleted. As with any environment, a best practice is to back up your data frequently, or whenever it's appropriate for your business continuity requirements.

Design your applications to gracefully handle instance termination

This topic covers features that you can use to prevent your Auto Scaling group from terminating instances that aren't yet ready to terminate, or from terminating instances too quickly for them to complete their assigned jobs. You can use all three of these features in combination or separately to design your applications to gracefully handle instance termination.

For example, suppose you have an Amazon SQS queue that collects incoming messages for long-running jobs. When a new message arrives, an instance in the Auto Scaling group retrieves the message and starts processing it. Each message takes 3 hours to process. As the number of messages increase, new instances are automatically added to the Auto Scaling group. As the number of messages decrease, existing instances are automatically terminated. In this case, Amazon EC2 Auto Scaling must decide which instance to terminate. By default, it's possible that Amazon EC2 Auto Scaling might terminate an instance that is 2.9 hours into processing a 3-hour long job, rather than an instance that's currently idle. To avoid issues with unexpected terminations when using Amazon EC2 Auto Scaling, you must design your application to respond to this scenario.

Contents

- Instance scale-in protection
- Custom termination policy
- Termination lifecycle hooks

Important

When designing your applications on Amazon EC2 Auto Scaling to gracefully handle instance termination, keep these points in mind.

- If an instance is unhealthy, Amazon EC2 Auto Scaling will replace it regardless of which feature you use (unless you suspend the `ReplaceUnhealthy` process). You can use a

lifecycle hook to allow the application to shut down gracefully or copy any data that you need to recover before the instance is terminated.

- A termination lifecycle hook is not guaranteed to run or finish before an instance is terminated. If something fails, Amazon EC2 Auto Scaling still terminates the instance.

Instance scale-in protection

You can use instance scale-in protection in many situations where terminating instances is a critical action that should be denied by default, and only explicitly allowed for specific instances. For example, when running containerized workloads, it’s common to want to protect all instances and remove protection only for instances with no current or scheduled tasks. Services such as Amazon ECS have built integrations with instance scale-in protection into their products.

You can enable scale-in protection on the Auto Scaling group to apply scale-in protection to instances when they're created and enable it for existing instances. When an instance has no more work to do, it can toggle off protection. The instance can continue polling for new jobs and re-enable protection when there are new jobs assigned.

Applications can set protection either from a centralized control plane that manages whether an instance is terminable or not, or from the instances themselves. However, a large fleet could run into throttling issues if large numbers of instances are continuously toggling their scale-in protection.

For more information, see Use instance scale-in protection to control instance termination.

Custom termination policy

Like instance scale-in protection, a custom termination policy helps you prevent your Auto Scaling group from terminating specific instances.

By default, your Auto Scaling group uses a default termination policy to determine which instances it terminates first. If you want more control over which instances terminate first, you can implement your own custom termination policy using a Lambda function. Amazon EC2 Auto Scaling calls the function whenever it must decide which instance to terminate. It will only terminate an instance that's returned by the function. If the function errors, times out, or produces an empty list, Amazon EC2 Auto Scaling doesn't terminate instances.

A custom termination policy is useful if it's known when an instance is sufficiently redundant or underutilized so that it can be terminated. To support this, you need to implement your application

with a control plane that monitors workload across the group. That way, if an instance is still processing jobs, the Lambda function knows not to include it.

For more information, see Create a custom termination policy with Lambda.

Termination lifecycle hooks

A termination lifecycle hook extends the life of an instance that's already selected for termination. It provides extra time to complete all messages or requests currently assigned to the instance, or to save progress and transfer the work to another instance.

For many workloads, a lifecycle hook may be enough to gracefully shut down an application on an instance that's selected for termination. This is a best-effort approach and can't be used to prevent termination if there's a failure.

To use a lifecycle hook, you need to know when an instance is selected to be terminated. You have two ways to know this:

Option	Description	Best used for	Link to documentation
Inside the instance	The Instance Metadata Service (IMDS) is a secure endpoint that you can poll for the status of an instance directly from the instance. If the metadata comes back with `Terminated` , then your instance is scheduled to be terminated.	Applications where you must perform an action on the instance before the instance is terminated.	Retrieve the target lifecycle state
Outside the instance	When an instance is terminating, an event notification is generated. You can create rules using Amazon EventBridge, Amazon SQS, or Amazon SNS to capture these events,	Applications that need to take action outside of the instance.	Configure a notification target

Option	Description	Best used for	Link to documentation
	and invoke a response such as with a Lambda function.		

To use a lifecycle hook, you also need to know when your instance is ready to be fully terminated. Amazon EC2 Auto Scaling will not tell Amazon EC2 to terminate the instance until it receives a CompleteLifecycleAction call or the timeout elapses, whichever happens first.

By default, an instance can continue running for one hour (heartbeat timeout) due to a termination lifecycle hook. You can configure the default timeout if one hour is not enough time to complete the lifecycle action. When a lifecycle action is actually in progress, you can extend the timeout with RecordLifecycleActionHeartbeat API calls.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks.

Suspend and resume Amazon EC2 Auto Scaling processes

This topic describes how to suspend and then resume one or more of the processes for your Auto Scaling group to temporarily disable certain operations.

Suspending processes can be useful when you need to investigate or troubleshoot an issue without interference from scaling policies or scheduled actions. It also helps prevent Amazon EC2 Auto Scaling from marking instances unhealthy and replacing them while you are making changes to your Auto Scaling group.

Topics

- Types of processes
- Considerations for suspending processes
- Suspend processes
- Resume processes
- How suspended processes affect other processes

> **Note**
>
> In addition to suspensions that you initiate, Amazon EC2 Auto Scaling can also suspend processes for Auto Scaling groups that repeatedly fail to launch instances. This is known as an *administrative suspension*. An administrative suspension most commonly applies to Auto Scaling groups that have been trying to launch instances for over 24 hours but have not succeeded in launching any instances. You can resume processes that were suspended by Amazon EC2 Auto Scaling for administrative reasons.

Types of processes

The suspend-resume feature supports the following processes:

- `Launch` – Adds instances to the Auto Scaling group when the group scales out, or when Amazon EC2 Auto Scaling chooses to launch instances for other reasons, such as when it adds instances to a warm pool.
- `Terminate` – Removes instances from the Auto Scaling group when the group scales in, or when Amazon EC2 Auto Scaling chooses to terminate instances for other reasons, such as when an instance is terminated for exceeding its maximum lifetime duration or failing a health check.
- `AddToLoadBalancer` – Adds instances to the attached load balancer target group or Classic Load Balancer when they are launched. For more information, see Use Elastic Load Balancing; to distribute incoming application traffic in your Auto Scaling group .
- `AlarmNotification` – Accepts notifications from CloudWatch alarms that are associated with dynamic scaling policies. For more information, see Dynamic scaling for Amazon EC2 Auto Scaling.
- `AZRebalance` – Balances the number of EC2 instances in the group evenly across all of the specified Availability Zones when the group becomes unbalanced, for example, when a previously unavailable Availability Zone returns to a healthy state. For more information, see Rebalancing activities.
- `HealthCheck` – Checks the health of the instances and marks an instance as unhealthy if Amazon EC2 or Elastic Load Balancing tells Amazon EC2 Auto Scaling that the instance is unhealthy. This process can override the health status of an instance that you set manually. For more information, see Health checks for instances in an Auto Scaling group.
- `InstanceRefresh` – Terminates and replaces instances using the instance refresh feature. For more information, see Use an instance refresh to update instances in an Auto Scaling group.

- `ReplaceUnhealthy` – Terminates instances that are marked as unhealthy and then creates new instances to replace them. For more information, see Health checks for instances in an Auto Scaling group.
- `ScheduledActions` – Performs the scheduled scaling actions that you create or that are created for you when you create an AWS Auto Scaling scaling plan and turn on predictive scaling. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling.

Considerations for suspending processes

Consider the following before suspending processes:

- Suspending `AlarmNotification` allows you to temporarily stop the group's target tracking, step, and simple scaling policies without deleting the scaling policies or their associated CloudWatch alarms. To temporarily stop individual scaling policies instead, see Disable a scaling policy for an Auto Scaling group.
- You might choose to suspend the `HealthCheck` and `ReplaceUnhealthy` processes to reboot instances without Amazon EC2 Auto Scaling terminating the instances based on its health checks. However, if you need Amazon EC2 Auto Scaling to continue performing health checks on the remaining instances, use the standby feature instead. For more information, see Temporarily remove instances from your Auto Scaling group.
- If you suspend the `Launch` and `Terminate` processes, or `AZRebalance`, and then you make changes to your Auto Scaling group, for example, by detaching instances or changing the Availability Zones that are specified, your group can become unbalanced between Availability Zones. If that happens, after you resume the suspended processes, Amazon EC2 Auto Scaling gradually redistributes instances evenly between the Availability Zones.
- If you suspend the `Terminate` process, you can still force instances to be terminated by using the delete-auto-scaling-group command with the force delete option.
- Suspending the `Terminate` process applies only to instances that are currently in the `InService` state. It does not prevent the termination of instances in other states, such as `Pending`, or instances that fail to resume properly from standby.
- The `RemoveFromLoadBalancerLowPriority` process can be ignored when it is present in calls to describe Auto Scaling groups using the AWS CLI or SDKs. This process is deprecated and is retained only for backward compatibility.

Suspend processes

To suspend a process for an Auto Scaling group, use one of the following methods:

Console

To suspend a process

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Details** tab, choose **Advanced configurations**, **Edit**.
4. For **Suspended processes**, choose the process to suspend.
5. Choose **Update**.

AWS CLI

Use the following suspend-processes command to suspend individual processes.

```
aws autoscaling suspend-processes --auto-scaling-group-name my-asg --scaling-
processes HealthCheck ReplaceUnhealthy
```

To suspend all processes, omit the `--scaling-processes` option, as follows.

```
aws autoscaling suspend-processes --auto-scaling-group-name my-asg
```

Resume processes

To resume a suspended process for an Auto Scaling group, use one of the following methods:

Console

To resume a suspended process

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.

2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Details** tab, choose **Advanced configurations**, **Edit**.
4. For **Suspended processes**, remove the suspended process.
5. Choose **Update**.

AWS CLI

To resume a suspended process, use the following resume-processes command.

```
aws autoscaling resume-processes --auto-scaling-group-name my-asg --scaling-
processes HealthCheck
```

To resume all suspended processes, omit the `--scaling-processes` option, as follows.

```
aws autoscaling resume-processes --auto-scaling-group-name my-asg
```

How suspended processes affect other processes

The following sections describe what happens when different processes are suspended individually.

Topics

- Launch is suspended
- Terminate is suspended
- AddToLoadBalancer is suspended
- AlarmNotification is suspended
- AZRebalance is suspended
- HealthCheck is suspended
- InstanceRefresh is suspended
- ReplaceUnhealthy is suspended
- ScheduledActions is suspended
- Additional considerations

Launch is suspended

- `AlarmNotification` is still active, but your Auto Scaling group can't initiate scale-out activities for alarms that are in breach.
- `ScheduledActions` is active, but your Auto Scaling group can't initiate scale-out activities for any scheduled actions that occur.
- `AZRebalance` stops rebalancing the group.
- `ReplaceUnhealthy` continues to terminate unhealthy instances, but does not launch replacements. When you resume the `Launch` process, Amazon EC2 Auto Scaling immediately replaces any instances that it terminated during the time that `Launch` was suspended.
- `InstanceRefresh` does not replace instances.

`Terminate` is suspended

- `AlarmNotification` is still active, but your Auto Scaling group can't initiate scale in activities for alarms that are in breach.
- `ScheduledActions` is active, but your Auto Scaling group can't initiate scale in activities for any scheduled actions that occur.
- `AZRebalance` is still active but does not function properly. It can launch new instances without terminating the old ones. This could cause your Auto Scaling group to grow up to 10 percent larger than its maximum size, because this is allowed temporarily during rebalancing activities. Your Auto Scaling group could remain above its maximum size until you resume the `Terminate` process.
- `ReplaceUnhealthy` is inactive but not `HealthCheck`. When `Terminate` resumes, the `ReplaceUnhealthy` process immediately starts running. If any instances were marked as unhealthy while `Terminate` was suspended, they are immediately replaced.
- `InstanceRefresh` does not replace instances.

`AddToLoadBalancer` is suspended

- Amazon EC2 Auto Scaling launches the instances but does not add them to the load balancer target group or Classic Load Balancer. When you resume the `AddToLoadBalancer` process, it resumes adding instances to the load balancer when they are launched. However, it does not

add the instances that were launched while this process was suspended. You must register those instances manually.

`AlarmNotification` is suspended

- Amazon EC2 Auto Scaling does not invoke scaling policies when a CloudWatch alarm threshold is in breach. When you resume `AlarmNotification`, Amazon EC2 Auto Scaling considers policies with alarm thresholds that are currently in breach.

`AZRebalance` is suspended

- Amazon EC2 Auto Scaling does not attempt to redistribute instances after certain events. However, if a scale-out or scale in event occurs, the scaling process still tries to balance the Availability Zones. For example, during scale out, it launches the instance in the Availability Zone with the fewest instances. If the group becomes unbalanced while `AZRebalance` is suspended and you resume it, Amazon EC2 Auto Scaling attempts to rebalance the group. It first calls `Launch` and then `Terminate`.

`HealthCheck` is suspended

- Amazon EC2 Auto Scaling stops marking instances unhealthy as a result of EC2 and Elastic Load Balancing health checks. Your custom health checks continue to function properly. After you suspend `HealthCheck`, if you need to, you can manually set the health state of instances in your group and have `ReplaceUnhealthy` replace them.

`InstanceRefresh` is suspended

- Amazon EC2 Auto Scaling stops replacing instances as a result of an instance refresh. If there is an instance refresh in progress, this pauses the operation without canceling it.

`ReplaceUnhealthy` is suspended

- Amazon EC2 Auto Scaling stops replacing instances that are marked as unhealthy. Instances that fail EC2 or Elastic Load Balancing health checks are still marked as unhealthy. As soon as you resume the `ReplaceUnhealthy` process, Amazon EC2 Auto Scaling replaces instances that were

marked unhealthy while this process was suspended. The `ReplaceUnhealthy` process calls `Terminate` first and then `Launch`.

`ScheduledActions` is suspended

- Amazon EC2 Auto Scaling does not run scheduled actions that are scheduled to run during the suspension period. When you resume `ScheduledActions`, Amazon EC2 Auto Scaling only considers scheduled actions whose scheduled time has not yet passed.

Additional considerations

In addition, when `Launch` or `Terminate` are suspended, the following features might not function correctly:

- **Maximum instance lifetime** – When `Launch` or `Terminate` are suspended, the maximum instance lifetime feature can't replace any instances.
- **Spot Instance interruptions** – If `Terminate` is suspended and your Auto Scaling group has Spot Instances, they can still terminate in the event that Spot capacity is no longer available. While `Launch` is suspended, Amazon EC2 Auto Scaling can't launch replacement instances from another Spot Instance pool or from the same Spot Instance pool when it is available again.
- **Capacity Rebalancing** – If `Terminate` is suspended and you use Capacity Rebalancing to handle Spot Instance interruptions, the Amazon EC2 Spot service can still terminate instances in the event that Spot capacity is no longer available. If `Launch` is suspended, Amazon EC2 Auto Scaling can't launch replacement instances from another Spot Instance pool or from the same Spot Instance pool when it is available again.
- **Attaching and detaching instances** – When `Launch` and `Terminate` are suspended, you can detach instances that are attached to your Auto Scaling group, but while `Launch` is suspended, you can't attach new instances to the group.
- **Standby instances** – When `Launch` and `Terminate` are suspended, you can put an instance in the `Standby` state, but while `Launch` is suspended, you can't return an instance in the `Standby` state to service.

Monitor your Amazon EC2 Auto Scaling groups

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon EC2 Auto Scaling and your AWS Cloud solutions. AWS provides the following monitoring tools to watch Amazon EC2 Auto Scaling, report when something is wrong, and take automatic actions when appropriate:

Health checks

Amazon EC2 Auto Scaling periodically performs health checks on the instances in your Auto Scaling group. If an instance does not pass its health check, it is marked unhealthy and will be terminated while Amazon EC2 Auto Scaling launches a new instance to replace it. For more information, see Health checks for instances in an Auto Scaling group.

AWS Health Dashboard

The AWS Health Dashboard displays information, and also provides notifications that are invoked by changes in the health of AWS resources. The information is presented in two ways: on a dashboard that shows recent and upcoming events organized by category, and in a full event log that shows all events from the past 90 days. For more information, see AWS Health Dashboard notifications for Amazon EC2 Auto Scaling.

CloudTrail

With AWS CloudTrail, you can track the calls made to the Amazon EC2 Auto Scaling API by or on behalf of your AWS account. CloudTrail stores the information in log files in the Amazon S3 bucket that you specify. You can use these log files to monitor activity of your Auto Scaling groups. Logs include which requests were made, the source IP addresses where the requests came from, who made the request, when the request was made, and so on. For more information, see Log Amazon EC2 Auto Scaling API calls with AWS CloudTrail.

> **Log collection for your Amazon EC2 instances**
>
> You can use CloudWatch to collect logs from the operating systems for your EC2 instances. For more information, see Collect metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent and View log data sent to CloudWatch Logs in the *Amazon CloudWatch User Guide*.

For information about other AWS services that can help you log and collect data about your workloads, see the Logging and monitoring guide for application owners guide in the *AWS Prescriptive Guidance*.

Amazon CloudWatch

Amazon CloudWatch helps you analyze logs and, in real time, monitor the metrics of your AWS resources and hosted applications. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can be notified when network activity is suddenly higher or lower than a metric's expected value. For more information about using this service to monitor the metrics of your Auto Scaling groups and instances, see Monitor CloudWatch metrics for your Auto Scaling groups and instances.

CloudWatch also tracks AWS API usage metrics for Amazon EC2 Auto Scaling. You can use these metrics to configure alarms that alert you when your API call volume violates a threshold that you define. For more information, see AWS usage metrics in the *Amazon CloudWatch User Guide*.

AWS Compute Optimizer

Compute Optimizer provides Amazon EC2 instance recommendations that can help you decide whether to move to a new instance type. It analyzes whether an Auto Scaling group's instance type is optimal and generates recommendations to reduce the cost and improve the performance of your workloads. For more information, see Get instance type recommendations with AWS Compute Optimizer.

Amazon EventBridge

Amazon EventBridge is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This lets you monitor events that happen in services, and build event-driven architectures. For more information, see Use EventBridge to handle Auto Scaling events.

AWS Security Hub

Use AWS Security Hub to monitor your usage of Amazon EC2 Auto Scaling as it relates to security best practices. Security Hub uses detective *security controls* to evaluate resource

configurations and *security standards* to help you comply with various compliance frameworks. For more information about using Security Hub to evaluate Amazon EC2 Auto Scaling resources, see Amazon EC2 Auto Scaling controls in the *AWS Security Hub User Guide*.

Amazon Simple Notification Service

You can configure Auto Scaling groups to send Amazon SNS notifications when Amazon EC2 Auto Scaling launches or terminates instances. For more information, see Amazon SNS notification options for Amazon EC2 Auto Scaling.

Health checks for instances in an Auto Scaling group

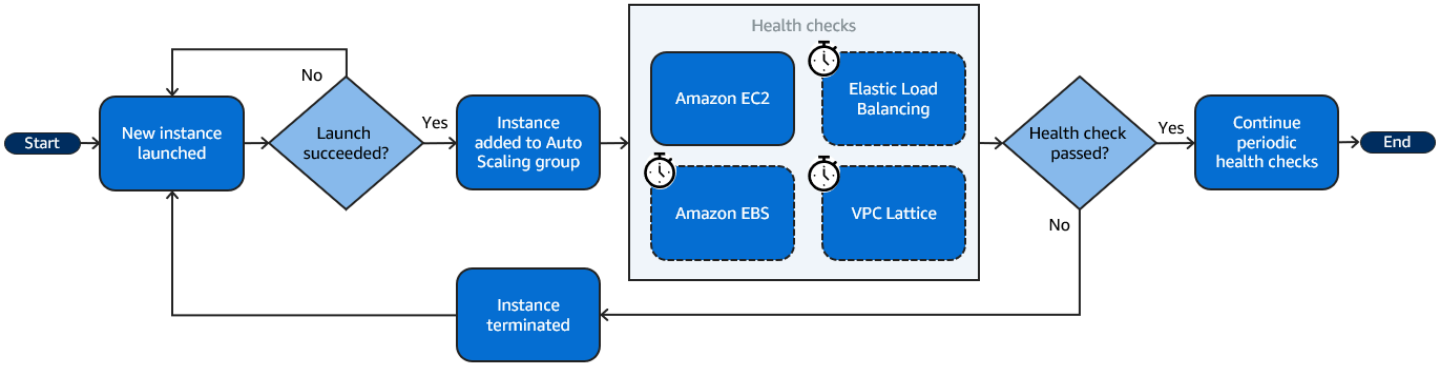
Amazon EC2 Auto Scaling continuously monitors the health status of instances in an Auto Scaling group to maintain the desired capacity.

All instances in an Auto Scaling group start with a `Healthy` status. Instances are assumed to be healthy unless Amazon EC2 Auto Scaling receives notification that they are unhealthy. It can receive notifications from various sources when an instance becomes unhealthy and must be replaced. These sources include the following:

- Amazon EC2
- Elastic Load Balancing
- VPC Lattice
- Amazon EBS
- Custom health checks that you define

When Amazon EC2 Auto Scaling determines that an `InService` instance is unhealthy, it replaces it with a new instance to maintain the desired capacity of the group. The new instance launches using the current settings of the Auto Scaling group and its associated launch template or launch configuration.

The following flow diagram illustrates the process of launching a new instance in an Auto Scaling group. It begins by launching the instance. If the launch succeeds, the instance gets added to the Auto Scaling group. Then, Amazon EC2 Auto Scaling performs health checks on the instance by using the built-in Amazon EC2 status checks, and after a grace period, any optional health checks that you enabled for the group. These health checks continue periodically. If any of the health checks fail, the instance is replaced.



Unhealthy instances can also occur when an instance terminates unexpectedly, such as from a Spot Instance interruption or manual termination by a user. Again, Amazon EC2 Auto Scaling will automatically launch a replacement instance in these cases to maintain the desired capacity.

Contents

- About the health checks for your Auto Scaling group
- Set the health check grace period for an Auto Scaling group
- Monitor Auto Scaling instances with impaired Amazon EBS volumes using health checks
- Set up a custom health check for your Auto Scaling group
- View the reason for health check failures
- Troubleshoot unhealthy instances in Amazon EC2 Auto Scaling

About the health checks for your Auto Scaling group

This topic provides an overview of the available health check types and describes the key considerations for integrating Amazon EC2 Auto Scaling health checks with your applications.

Contents

- Health check types
- Amazon EC2 health checks
- Elastic Load Balancing health checks
- VPC Lattice health checks
- How Amazon EC2 Auto Scaling minimizes downtime
- Health checks for instances in a warm pool
- Health check considerations

Health check types

Amazon EC2 Auto Scaling can determine the health status of an `InService` instance by using one or more of the following health checks:

Health check type	What it checks
Amazon EC2 status checks and scheduled events	• Checks that the instance is running. • Checks for underlying hardware or software issues that might impair the instance. This is the default health check type for an Auto Scaling group.
Elastic Load Balancing health checks	• Checks whether the load balancer reports the instance as healthy, confirming whether the instance is available to handle requests. To run this health check type, you must turn it on for your Auto Scaling group.
VPC Lattice health checks	• Checks whether VPC Lattice reports the instance as healthy, confirming whether the instance is available to handle requests. To run this health check type, you must turn it on for your Auto Scaling group.
Amazon EBS health checks	• Checks whether EBS volumes are reachable and passing I/O status checks. To run this health check type, you must turn it on for your Auto Scaling group.
Custom health checks	• Checks for any other problems that might indicate instance health issues, according to your custom health checks.

Amazon EC2 health checks

After an instance launches, it's attached to the Auto Scaling group and enters the `InService` state. For more information about the different lifecycle states for instances in an Auto Scaling group, see Amazon EC2 Auto Scaling instance lifecycle.

Amazon EC2 Auto Scaling periodically checks the health status of all instances within the Auto Scaling group to make sure that they're running and in good condition.

Status checks

Amazon EC2 Auto Scaling uses the results of the Amazon EC2 instance status checks and system status checks to determine the health status of an instance. If the instance is in any Amazon EC2 state other than `running`, or if its status for the status checks becomes `impaired`, Amazon EC2 Auto Scaling considers the instance to be unhealthy and replaces it. This includes when the instance has any of the following states:

- `stopping`
- `stopped`
- `shutting-down`
- `terminated`

The Amazon EC2 status checks do not require any special configuration and are always enabled. For more information, see Types of status checks in the *Amazon EC2 User Guide*.

> **Important**
>
> Amazon EC2 Auto Scaling lets the status checks fail occasionally, without taking any action. When a status check fails, Amazon EC2 Auto Scaling waits a few minutes for AWS to fix the issue. It does not immediately mark an instance `Unhealthy` when its status for the status checks becomes `impaired`.
> However, if Amazon EC2 Auto Scaling detects that an instance is no longer in the `running` state, this situation is treated as an immediate failure. In this case, it immediately marks the instance as `Unhealthy` and replaces it.

Scheduled events

Amazon EC2 can occasionally schedule events on your instances to be run after a particular timestamp. For more information, see Scheduled events for your instances in the *Amazon EC2 User Guide*.

If one of your instances is affected by a scheduled event, Amazon EC2 Auto Scaling considers the instance to be unhealthy and replaces it. The instance doesn't start shutting down until the date and time that's specified in the timestamp is reached.

Elastic Load Balancing health checks

When you turn on Elastic Load Balancing health checks for your Auto Scaling group, Amazon EC2 Auto Scaling can use the results of those health checks to determine the health status of an instance.

Before you can turn on Elastic Load Balancing health checks for your Auto Scaling group, you must configure an Elastic Load Balancing load balancer and configure a health check for it to determine if your instances are healthy. For more information, see Prepare to attach an Elastic Load Balancing load balancer.

After you attach the load balancer to your Auto Scaling group, the following occurs:

- Amazon EC2 Auto Scaling registers the instances in the Auto Scaling group with the load balancer.
- After an instance finishes registering, it enters the `InService` state and becomes available for use with the load balancer.

By default, Amazon EC2 Auto Scaling ignores the results of the Elastic Load Balancing health checks. After you turn on these health checks for your Auto Scaling group, when Elastic Load Balancing reports a registered instance as `Unhealthy`, Amazon EC2 Auto Scaling marks the instance `Unhealthy` on its next periodic health check and replaces it.

If connection draining (deregistration delay) is enabled for your load balancer, Amazon EC2 Auto Scaling waits for either in-flight requests to complete or the maximum timeout to expire before it terminates unhealthy instances.

> **Note**
>
> For instructions for how to attach the load balancer and turn on Elastic Load Balancing health checks for your Auto Scaling group, see Attach an Elastic Load Balancing load balancer to your Auto Scaling group.
> When you turn on Elastic Load Balancing health checks for a group, Amazon EC2 Auto Scaling can replace instances that Elastic Load Balancing reports as unhealthy, but only after the load balancer is in the `InService` state. For more information, see Verify the attachment status of your load balancer.

VPC Lattice health checks

By default, Amazon EC2 Auto Scaling ignores the results of the VPC Lattice health checks. You can optionally turn on these health checks for your Auto Scaling group. After you do this, when VPC Lattice reports a registered instance as `Unhealthy`, Amazon EC2 Auto Scaling marks the instance `Unhealthy` on its next periodic health check and replaces it. The process of registering instances and then checking their health is the same as how Elastic Load Balancing health checks work.

> **Note**
>
> For instructions for how to attach the VPC Lattice target group and turn on VPC Lattice health checks for your Auto Scaling group, see Attach a VPC Lattice target group to your Auto Scaling group.
> When you turn on VPC Lattice health checks for a group, Amazon EC2 Auto Scaling can replace instances that VPC Lattice reports as unhealthy, but only after the target group is in the `InService` state. For more information, see Verify the attachment status of your VPC Lattice target group.

How Amazon EC2 Auto Scaling minimizes downtime

By default, new instances are provisioned at the same time your existing instances are terminated, which might prevent new requests from being accepted until the new instances are fully operational.

If Amazon EC2 Auto Scaling determines that any instances are no longer running (or they were marked `Unhealthy` with the set-instance-health command), it immediately replaces them.

However, if other instances are found to be unhealthy, Amazon EC2 Auto Scaling uses the following approach to recover from failures. This approach minimizes any downtime that might occur because of temporary issues or misconfigured health checks.

- If a scaling activity is in progress and your Auto Scaling group is less than its desired capacity by 10 percent or more, Amazon EC2 Auto Scaling waits for the in-progress scaling activity before replacing the unhealthy instances.
- When scaling out, Amazon EC2 Auto Scaling waits for the instances to pass an initial health check. It also waits for the default instance warmup to finish to make sure that the new instances are ready.
- After the instances finish warming up and the group has risen to more than 90 percent of its desired capacity, Amazon EC2 Auto Scaling replaces the unhealthy instances as follows:
  - Amazon EC2 Auto Scaling only replaces up to 10 percent of the group's desired capacity at a time. It does this until all of the unhealthy instances are replaced.
  - When replacing instances, it waits for the new instances to pass an initial health check. It also waits for the default instance warmup to finish before continuing.

> **Note**
>
> If the size of an Auto Scaling group is small enough that the resulting value of 10 percent is less than one, Amazon EC2 Auto Scaling instead replaces the unhealthy instances one at a time. This might result in some downtime for the group.
> Also, if all instances in an Auto Scaling group are reported unhealthy by Elastic Load Balancing health checks and the load balancer is in the `InService` state, Amazon EC2 Auto Scaling might mark fewer instances unhealthy at a time. This can result in much fewer instances replaced at a time than the 10 percent applied in other scenarios. This provides you with time to fix the problem without Amazon EC2 Auto Scaling automatically terminating the entire group.

Health checks for instances in a warm pool

Amazon EC2 Auto Scaling also performs health checks on instances in a warm pool. For more information, see View health check status and the reason for health check failures.

Health check considerations

The following are considerations when using Amazon EC2 Auto Scaling health checks.

- If you need something to happen on the instance that is terminating, or on the instance that is starting up, you can use lifecycle hooks. These hooks let you perform a custom action as Amazon EC2 Auto Scaling launches or terminates instances. For more information, see Amazon EC2 Auto Scaling lifecycle hooks.
- Amazon EC2 Auto Scaling does not provide a way of removing the Amazon EC2 status checks and scheduled events from its health checks. If you do not want instances to be replaced, we recommend that you suspend the `ReplaceUnhealthy` and `HealthCheck` process for individual Auto Scaling groups. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.
- To manually set an unhealthy instance's health status back to `Healthy`, you can try to use the set-instance-health command. If you get an error, this is probably because the instance is already terminating. Generally, setting an instance's health status back to `Healthy` with the set-instance-health command is only useful in cases where either the `ReplaceUnhealthy` process or the `Terminate` process is suspended.
- If you need to troubleshoot an instance without interference from health checks, you can put the instance in `Standby` state. Amazon EC2 Auto Scaling does not perform health checks on instances that are in the `Standby` state until you put the instances back in service. For more information, see Temporarily remove instances from your Auto Scaling group.
- When your instance is terminated, any associated Elastic IP addresses are disassociated and are not automatically associated with the new instance. You must manually associate the Elastic IP addresses with the new instance, or do it automatically with a lifecycle hook-based solution. For more information, see Elastic IP addresses in the *Amazon EC2 User Guide*.
- Similarly, when your instance is terminated, its attached EBS volumes are detached (or deleted depending on the volume's `DeleteOnTermination` attribute). You must manually attach these EBS volumes to the new instance, or do it automatically with a lifecycle hook-based solution. For more information, see Attach an Amazon EBS volume to an instance in the *Amazon EBS User Guide*.

Set the health check grace period for an Auto Scaling group

When an Amazon EC2 Auto Scaling health check determines that an `InService` instance is unhealthy, it replaces it with a new instance. The health check grace period specifies the minimum

amount of time (in seconds) to keep a new instance in service before terminating it if it's found to be unhealthy.

An example use case might be a requirement for Amazon EC2 Auto Scaling to avoid taking action if the Elastic Load Balancing health checks fail and the cause is that the instance is still initializing. Elastic Load Balancing health checks run in parallel, starting when the instance is registered with the load balancer. The grace period prevents Amazon EC2 Auto Scaling from marking your newly launched instances `Unhealthy` and terminating them unnecessarily if they don't immediately pass these health checks after they enter the `InService` state.

In the console, by default, the health check grace period is 300 seconds when you create an Auto Scaling group. Its default value is 0 seconds when you create an Auto Scaling group using the AWS CLI or an SDK. A value of 0 turns off the health check grace period.

Setting this value too high reduces the effectiveness of the Amazon EC2 Auto Scaling health checks. If you use lifecycle hooks for instance launch, you can set the health check grace period to 0. With lifecycle hooks, Amazon EC2 Auto Scaling provides a way to make sure that instances are always initialized before they enter the `InService` state. For more information, see Amazon EC2 Auto Scaling lifecycle hooks.

The grace period applies to the following instances:

- Newly launched instances
- Instances that are put back into service after being in standby
- Instances that you manually attach to the group

> **Important**
>
> During the health check grace period, if Amazon EC2 Auto Scaling detects that an instance is no longer in the Amazon EC2 `running` state, it immediately marks the instance `Unhealthy` and replaces it. For example, if you stop an instance in an Auto Scaling group, it is marked `Unhealthy` and replaced.

Set the health check grace period for a group

You can set the health check grace period for new and existing Auto Scaling groups.

Console

To modify the health check grace period for a new group

When you create the Auto Scaling group, enter the amount of time (in seconds) on the **Configure advanced options** page, **Health checks**, **Health check grace period**. This is how long Amazon EC2 Auto Scaling must wait before checking the health status of an instance after it enters the `InService` state.

AWS CLI

To modify the health check grace period for a new group

Add the `--health-check-grace-period` option to the create-auto-scaling-group command. The following example configures the health check grace period with a value of *60* seconds for a new Auto Scaling group named *my-asg*.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-grace-period 60 ...
```

Console

To modify the health check grace period for an existing group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your Auto Scaling group in.
3. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the page.
4. On the **Details** tab, choose **Health checks**, **Edit**.
5. Under **Health check grace period**, enter the amount of time, in seconds. This is how long Amazon EC2 Auto Scaling must wait before checking the health status of an instance after it enters the `InService` state.
6. Choose **Update**.

AWS CLI

To modify the health check grace period for an existing group

Add the `--health-check-grace-period` option to the update-auto-scaling-group command. The following example configures the health check grace period with a value of *120* seconds for an existing Auto Scaling group named *my-asg*.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-grace-period 120
```

> **Note**
>
> We strongly recommend also setting the default instance warm-up time for your Auto Scaling group. For more information, see Set the default instance warmup for an Auto Scaling group.

Monitor Auto Scaling instances with impaired Amazon EBS volumes using health checks

You can turn on the Amazon EBS health checks for your Auto Scaling group to make sure that Amazon EC2 Auto Scaling monitors the entire system on which your application runs.

After you turn on these health checks, Amazon EC2 Auto Scaling receives the results of the Amazon EC2 status checks performed on an instance's attached EBS volumes. If a volume is not reachable or does not pass I/O status checks, the health check will fail, and the corresponding instance will be considered unhealthy. When Amazon EC2 Auto Scaling detects an unhealthy instance, it replaces it.

This topic assumes you're familiar with the attached EBS status checks. If you're not, see the Attached EBS status checks section of the *Amazon EC2 User Guide* for details. The following topic describes how to turn on the Amazon EC2 Auto Scaling health checks that rely on the attached EBS status checks.

> **Note**
>
> You can turn on the Amazon EBS health checks for all of your Auto Scaling groups. However, these health checks are only available for instances built on the AWS Nitro System.

Turn on the Amazon EBS health checks for a group

You can turn on the Amazon EBS health checks for new and existing Auto Scaling groups.

Console

Turning on Amazon EBS health checks for a new group

When you create the Auto Scaling group, on the **Configure advanced options** page, for **Health checks**, **Additional health check types**, select **Turn on Amazon EBS health checks**. Then, for **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling must wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.

AWS CLI

Turning on Amazon EBS health checks for a new group

Add the `--health-check-type` option to the create-auto-scaling-group command. The following example specifies **EBS** for the `--health-check-type` option for a new Auto Scaling group named *my-asg*.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "EBS" --health-check-grace-period 60 ...
```

You can specify multiple values for the `--health-check-type` option. For example, to add both Amazon EBS and Elastic Load Balancing health checks types, use the following command.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "EBS,ELB" --health-check-grace-period 60 ...
```

Value names are case sensitive.

Console

Turning on Amazon EBS health checks for an existing group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your Auto Scaling group in.
3. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
4. On the **Details** tab, choose **Health checks**, **Edit**.
5. For **Health checks**, **Additional health check types**, select **Turn on Amazon EBS health checks**.
6. For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling must wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.
7. Choose **Update**.

AWS CLI

Turning on Amazon EBS health checks for an existing group

Add the `--health-check-type` option to the update-auto-scaling-group command. The following example specifies **EBS** for the `--health-check-type` option for an existing Auto Scaling group named *my-asg*.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "EBS" --health-check-grace-period 60
```

To use multiple health checks types, you can specify multiple values (for example, **EBS,ELB**) for the `--health-check-type` option.

Value names are case sensitive.

Turn off the Amazon EBS health checks for an Auto Scaling group

The following topic describes how to turn off Amazon EBS health checks for an Auto Scaling group. If you don't require Amazon EBS health checks anymore, use the following procedure to turn them off.

Console

Turning off Amazon EBS health checks for a group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Health checks**, **Edit**.
4. For **Health checks**, **Additional health check types**, deselect **Turn on Amazon EBS health checks**.
5. Choose **Update**.

AWS CLI

Turning off Amazon EBS health checks for a group

To update the health checks on an Auto Scaling group so that it no longer uses Amazon EBS health checks, use the update-auto-scaling-group command. Include the `--health-check-type` option and a value of **EC2**.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "EC2"
```

To turn off Amazon EBS health checks without turning off other health check types (such as Elastic Load Balancing), you must specify them instead of **EC2**. For example, for Elastic Load Balancing health checks, specify **ELB** for the `--health-check-type` option.

Value names are case sensitive.

Set up a custom health check for your Auto Scaling group

You can use custom health checks to complement the existing health check options provided by Amazon EC2 Auto Scaling. By combining custom health checks with the other health check types, you can create a comprehensive health monitoring system tailored to your application's needs.

To get started, create custom tests to verify that the instances in your Auto Scaling group are working correctly and can handle incoming traffic. If the health check that you configure detects that an instance isn't responding, then mark that particular instance as `Unhealthy`, which causes Amazon EC2 Auto Scaling to immediately replace it.

You can send the health status of an instance directly to Amazon EC2 Auto Scaling by using the AWS CLI or an SDK. The following examples show you how to use the AWS CLI to configure the health status of an instance and then verify the instance's health status.

Use the following set-instance-health command to set the health status of the specified instance to **`Unhealthy`**.

```
aws autoscaling set-instance-health --instance-id i-1234567890abcdef0 --health-
status Unhealthy
```

By default, this command respects the health check grace period. However, you can override this behavior and not respect the grace period by including the `--no-should-respect-grace-period` option.

Use the following describe-auto-scaling-groups command to verify that the instance's health status is `Unhealthy`.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names my-asg
```

The following is an example response that shows you that the health status of the instance is `Unhealthy`, and that the instance is terminating.

```
{
    "AutoScalingGroups": [
        {
            ....
            "Instances": [
                {
                    "ProtectedFromScaleIn": false,
                    "AvailabilityZone": "us-west-2a",
```

```
                    "LaunchTemplate": {
                        "LaunchTemplateName": "my-launch-template",
                        "Version": "1",
                        "LaunchTemplateId": "lt-1234567890abcdef0"
                    },
                    "InstanceId": "i-1234567890abcdef0",
                    "InstanceType": "t2.micro",
                    "HealthStatus": "Unhealthy",
                    "LifecycleState": "Terminating"
                },
                ...
            ]
        }
    ]
}
```

View the reason for health check failures

Using the following procedure, you can view information about any instances replaced due to a health check.

By default, Amazon EC2 Auto Scaling creates a new scaling activity for terminating the unhealthy instance and then terminates it. While the instance is terminating, another scaling activity launches a new instance. You can change this behavior to start launching a new instance as soon as possible by using an instance maintenance policy. For more information, see Instance maintenance policies.

Console

Viewing the reason for health check failures

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Activity** tab, under **Activity history**, the **Status** column shows whether your Auto Scaling group has successfully launched or terminated instances.

   If it terminated any unhealthy instances, the **Cause** column shows the date and time of the termination and the reason for the health check failure. For example, `At 2022-05-14T20:11:53Z an instance was taken out of service in response`

`to a user health-check`. This message indicates that a custom health check marked the instance unhealthy.

For help with health check failures, see Troubleshoot unhealthy instances in Amazon EC2 Auto Scaling.

AWS CLI

Viewing the reason for health check failures

Use the following describe-scaling-activities command.

```
aws autoscaling describe-scaling-activities --auto-scaling-group-name my-asg
```

The following is an example response, where `Cause` contains the reason for the health check failure.

```
{
  "Activities": [
    {
      "ActivityId": "4c65e23d-a35a-4e7d-b6e4-2eaa8753dc12",
      "AutoScalingGroupName": "my-asg",
      "Description": "Terminating EC2 instance: i-04925c838b6438f14",
      "Cause": "At 2021-04-01T21:48:35Z an instance was taken out of service in
 response to a user health-check.",
      "StartTime": "2021-04-01T21:48:35.859Z",
      "EndTime": "2021-04-01T21:49:18Z",
      "StatusCode": "Successful",
      "Progress": 100,
      "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":\"us-
west-2a\"...}",
      "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:283179a2-
f3ce-423d-93f6-66bb518232f7:autoScalingGroupName/my-asg"
    },
...
  ]
}
```

For a description of the fields in the output, see Activity in the *Amazon EC2 Auto Scaling API Reference*.

To describe the scaling activities after the Auto Scaling group has been deleted, add the `--include-deleted-groups` option to the describe-scaling-activities command.

Troubleshoot unhealthy instances in Amazon EC2 Auto Scaling

The following are error messages returned by Amazon EC2 Auto Scaling, the potential causes, and the steps you can take to resolve the issues.

To retrieve an error message, see View the reason for health check failures.

Error messages

- An instance was taken out of service in response to an EC2 instance status check failure
- An instance was taken out of service in response to an EC2 health check that indicated it had been terminated or stopped
- An instance was taken out of service in response to an ELB system health check failure
- Additional resources

An instance was taken out of service in response to an EC2 instance status check failure

Problem: Auto Scaling instances fail the Amazon EC2 status checks.

Cause 1: If there are issues that cause Amazon EC2 to consider the instances in your Auto Scaling group impaired, Amazon EC2 Auto Scaling automatically replaces the instances as part of its health checks.

Solution 1: When an instance status check fails, you typically must address the problem yourself by making instance configuration changes until your application is no longer exhibiting any problems. To resolve this issue, follow these steps:

1. Manually create an Amazon EC2 instance that is not part of the Auto Scaling group and investigate the problem. For general help with investigating impaired instances, see Troubleshoot instances with failed status checks in the *Amazon EC2 User Guide* and Troubleshooting Windows Instances in the *Amazon EC2 User Guide*.
2. After you confirm that your instance launched successfully and is healthy, deploy a new, error-free instance configuration to the Auto Scaling group.
3. Delete the instance that you created to avoid ongoing charges to your AWS account.

An instance was taken out of service in response to an EC2 health check that indicated it had been terminated or stopped

Problem: Auto Scaling instances that have been stopped, rebooted, or terminated are replaced.

Cause 1: A user manually stopped, rebooted, or terminated the instance.

Solution 1: If you need to stop or reboot the instances in your Auto Scaling group, we recommend that you put the instances on standby first. For more information, see Temporarily remove instances from your Auto Scaling group.

Cause 2: Amazon EC2 Auto Scaling attempts to replace Spot Instances after the Amazon EC2 Spot service interrupts the instances, because the Spot price increases above your maximum price or capacity is no longer available.

Solution 2: There is no guarantee that a Spot Instance exists to fulfill the request at any given point in time. However, you can try the following:

- Use a higher Spot maximum price (possibly the On-Demand price). By setting your maximum price higher, it gives the Amazon EC2 Spot service a better chance of launching and maintaining your required amount of capacity.
- Increase the number of different capacity pools that you can launch instances from by running multiple instance types in multiple Availability Zones. For more information, see Auto Scaling groups with multiple instance types and purchase options.
- If you use multiple instance types, consider enabling the Capacity Rebalancing feature. This is useful if you want the Amazon EC2 Spot service to attempt to launch a new Spot Instance before a running instance is terminated. For more information, see Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions.

Cause 3: With Capacity Blocks, Amazon EC2 terminates any instances that are still running 30 minutes before the end time of the Capacity Block. This abrupt termination causes your Auto Scaling group to try to launch new instances to maintain its desired capacity, even as the Capacity Block is ending.

Solution 3: To resolve this issue, try the following:

- Decrease the desired capacity of the Auto Scaling group to prevent it from trying to launch new instances. For more information, see Manual scaling for Amazon EC2 Auto Scaling.

- Make sure you scale in your Auto Scaling group 30 minutes before the Capacity Block end time so that you do not encounter this error frequently. Make sure any lifecycle hooks have completed 30 minutes before the Capacity Block end time. For more information, see Use Capacity Blocks for machine learning workloads.

An instance was taken out of service in response to an ELB system health check failure

Problem: Auto Scaling instances might pass the EC2 status checks. But they might fail the Elastic Load Balancing health checks for the target groups or Classic Load Balancers with which the Auto Scaling group is registered.

Cause 1: If your Auto Scaling group relies on health checks provided by Elastic Load Balancing, Amazon EC2 Auto Scaling determines the health status of your instances by checking the results of both the EC2 status checks and the Elastic Load Balancing health checks. The load balancer performs health checks by sending a request to each instance and waiting for the correct response, or by establishing a connection with the instance. An instance might fail the Elastic Load Balancing health check because an application running on the instance has issues that cause the load balancer to consider the instance out of service.

Solution 1: To pass the Elastic Load Balancing health checks:

- Verify that the health check settings of your target groups are correctly configured. You define health check settings for your load balancer per target group. For more information, see Configure health checks for targets.
- Make note of the success codes that the load balancer is expecting, and verify that your application is configured correctly to return these codes on success.
- Verify that the security groups for your load balancer and Auto Scaling group are correctly configured.
- Verify that the load balancer is configured in the same Availability Zones as your Auto Scaling group.

Solution 2: Update the Auto Scaling group to disable Elastic Load Balancing health checks. For instructions for how to disable these health checks, see Attach an Elastic Load Balancing load balancer to your Auto Scaling group.

Cause 2: There is a mismatch between the health check grace period and the instance startup time.

Solution 3: Edit the health check grace period for your Auto Scaling group. Set the grace period to a long enough time period to support the number of consecutive successful health checks required before Elastic Load Balancing considers a newly launched instance healthy. For more information, see Set the health check grace period for an Auto Scaling group.

Additional resources

If you have a different issue, see the following AWS re:Post articles for additional troubleshooting help:

- Why did Amazon EC2 Auto Scaling terminate an instance?
- Why didn't Amazon EC2 Auto Scaling terminate an unhealthy instance?

AWS Health Dashboard notifications for Amazon EC2 Auto Scaling

Your AWS Health Dashboard provides support for notifications that come from Amazon EC2 Auto Scaling. These notifications provide awareness and remediation guidance for resource performance or availability issues that may affect your applications. Only events that are specific to missing security groups and launch templates are currently available.

The AWS Health Dashboard is part of the AWS Health service. It requires no set up and can be viewed by any user that is authenticated in your account. For more information, see Getting started with your AWS Health Dashboard.

If you receive a message similar to the following messages, it should be treated as an alarm to take action.

Example: Auto Scaling group is not scaling out due to a missing security group

```
 Hello,

 At 2020-01-11 04:00 UTC, we detected an issue with your Auto Scaling group [ARN] in
 AWS account 123456789012.

 A security group associated with this Auto Scaling group cannot be found. Each time
a
 scale out operation is performed, it will be prevented until you make a change that
 fixes the issue.
```

```
 We recommend that you review and update your Auto Scaling group configuration to
change
 the launch template or launch configuration that depends on the unavailable security
 group.

 Sincerely,
 Amazon Web Services
```

Example: Auto Scaling group is not scaling out due to a missing launch template

```
 Hello,

 At 2021-05-11 04:00 UTC, we detected an issue with your Auto Scaling group [ARN] in
 AWS account 123456789012.

 The launch template associated with this Auto Scaling group cannot be found. Each
time
 a scale out operation is performed, it will be prevented until you make a change
that
 fixes the issue.

 We recommend that you review and update your Auto Scaling group configuration and
 specify an existing launch template to use.

 Sincerely,
 Amazon Web Services
```

Monitor CloudWatch metrics for your Auto Scaling groups and instances

Metrics are the fundamental concept in Amazon CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Think of a metric as a variable to monitor, and the data points as representing the values of that variable over time. You can use these metrics to verify that your system is performing as expected.

Amazon EC2 Auto Scaling metrics that collect information about Auto Scaling groups are in the `AWS/AutoScaling` namespace. Amazon EC2 instance metrics that collect CPU and other usage data from Auto Scaling instances are in the `AWS/EC2` namespace.

The Amazon EC2 Auto Scaling console displays a series of graphs for the group metrics and the aggregated instance metrics for the group. Depending on your needs, you might prefer to access

data for your Auto Scaling groups and instances from Amazon CloudWatch instead of the Amazon EC2 Auto Scaling console.

For more information, see the Amazon CloudWatch User Guide.

Contents

- View monitoring graphs in the Amazon EC2 Auto Scaling console
- Amazon CloudWatch metrics for Amazon EC2 Auto Scaling
- Configure monitoring for Auto Scaling instances

View monitoring graphs in the Amazon EC2 Auto Scaling console

In the Amazon EC2 Auto Scaling section of the Amazon EC2 console, you can monitor minute-by-minute progress of individual Auto Scaling groups using CloudWatch metrics.

You can monitor the following types of metrics:

- **Auto Scaling metrics** – Auto Scaling metrics are turned on only when you enable them. For more information, see Enable Auto Scaling group metrics (console). When Auto Scaling metrics are enabled, the monitoring graphs show data published at one-minute granularity for Auto Scaling metrics.
- **EC2 metrics** – The Amazon EC2 instance metrics are always enabled. When detailed monitoring is enabled, the monitoring graphs show data published at one-minute granularity for instance metrics. For more information, see Configure monitoring for Auto Scaling instances.

To view monitoring graphs using the Amazon EC2 Auto Scaling console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to the Auto Scaling group that you want to view metrics for.

   A split pane opens up in the bottom part of the **Auto Scaling groups** page.
3. Choose the **Monitoring** tab.

   Amazon EC2 Auto Scaling displays monitoring graphs for **Auto Scaling** metrics.
4. To view monitoring graphs of the aggregated instance metrics for the group, choose **EC2**.

Graph actions

- Hover on a data point to view a data pop-up for a specific time in UTC.
- To enlarge a graph, choose **Enlarge** from the menu tool (the three vertical dots) in the upper right of the graph. Alternatively, choose the maximize icon at the top of the graph.
- Adjust the time period for data displayed in the graph by selecting one of the predefined time period values. If the graph is enlarged, you can choose **Custom** to define your own time period.
- Choose **Refresh** from the menu tool to update the data in a graph.
- Drag your cursor over the graph data to select a specific range. You can then choose **Apply time range** in the menu tool.
- Choose **View logs** from the menu tool to view associated log streams (if any) in the CloudWatch console.
- To view a graph in CloudWatch, choose **View in metrics** from the menu tool. This takes you to the CloudWatch page for that graph. There, you can view more information or access historical information to gain a better understanding of how your Auto Scaling group changed over an extended period.

Graph metrics for your Auto Scaling groups

After you create an Auto Scaling group, you can open the Amazon EC2 Auto Scaling console and view the monitoring graphs for the group on the **Monitoring** tab.

In the **Auto Scaling** section, the graph metrics include the following metrics. These metrics provide measurements that can be indicators of a potential issue, such as number of terminating instances or number of pending instances. You can find definitions for these metrics in Amazon CloudWatch metrics for Amazon EC2 Auto Scaling.

Display name	CloudWatch metric name
Minimum Group Size	`GroupMinSize`
Maximum Group Size	`GroupMaxSize`
Desired Capacity	`GroupDesiredCapacity`
In Service Instances	`GroupInServiceInstances`

Display name	CloudWatch metric name
Pending Instances	`GroupPendingInstances`
Standby Instances	`GroupStandbyInstances`
Terminating Instances	`GroupTerminatingInstances`
Total Instances	`GroupTotalInstances`

In the **EC2** section, you can find the following graph metrics based on key performance metrics for your Amazon EC2 instances. These EC2 metrics are an aggregate of metrics for all instances in the group. You can find definitions for these metrics in List the available CloudWatch metrics for your instances in the *Amazon EC2 User Guide*.

Display name	CloudWatch metric name
CPU Utilization	`CPUUtilization`
Disk Reads	`DiskReadBytes`
Disk Read Operations	`DiskReadOps`
Disk Writes	`DiskWriteBytes`
Disk Write Operations	`DiskWriteOps`
Network In	`NetworkIn`
Network Out	`NetworkOut`
Status Check Failed (Any)	`StatusCheckFailed`
Status Check Failed (Instance)	`StatusCheckFailed_Instance`
Status Check Failed (System)	`StatusCheckFailed_System`

In addition, some metrics are available for specific use cases in the **Auto Scaling** graph metrics.

The following metrics are useful if your group uses weights that define how many units each instance contributes to the desired capacity of the group. You can find definitions for these metrics in Amazon CloudWatch metrics for Amazon EC2 Auto Scaling.

Display name	CloudWatch metric name
In Service Capacity Units	`GroupInServiceCapacity`
Pending Capacity Units	`GroupPendingCapacity`
Standby Capacity Units	`GroupStandbyCapacity`
Terminating Capacity Units	`GroupTerminatingCapacity`
Total Capacity Units	`GroupTotalCapacity`

The following metrics are useful if your group uses the warm pool feature. You can find definitions for these metrics in Amazon CloudWatch metrics for Amazon EC2 Auto Scaling.

Display name	CloudWatch metric name
Warm Pool Minimum Size	`WarmPoolMinSize`
Warm Pool Desired Capacity	`WarmPoolDesiredCapacity`
Warm Pool Pending Capacity Units	`WarmPoolPendingCapacity`
Warm Pool Terminating Capacity Units	`WarmPoolTerminatingCapacity`
Warm Pool Warmed Capacity Units	`WarmPoolWarmedCapacity`
Warm Pool Total Capacity Units Launched	`WarmPoolTotalCapacity`
Group and Warm Pool Desired Capacity	`GroupAndWarmPoolDesiredCapacity`

Display name	CloudWatch metric name
Group and Warm Pool Total Capacity Units Launched	`GroupAndWarmPoolTotalCapacity`

Related resources

- To monitor per-instance metrics, see Graph metrics for your instances in the *Amazon EC2 User Guide*.
- CloudWatch dashboards are customizable home pages in the CloudWatch console. You can use these pages to monitor your resources in a single view, even including resources that are spread across different Regions. You can use CloudWatch dashboards to create customized views of the metrics and alarms for your AWS resources. For more information, see the Amazon CloudWatch User Guide.

Amazon CloudWatch metrics for Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling publishes the following metrics in the `AWS/AutoScaling` namespace. The actual Auto Scaling group metrics made available will depend on whether you have group metrics enabled, and which group metrics you enabled. Group metrics are available at one-minute granularity at no additional charge, but you must enable them.

When you enable Auto Scaling group metrics, Amazon EC2 Auto Scaling sends sampled data to CloudWatch every minute on a best-effort basis. In rare cases when CloudWatch experiences a service disruption, data isn't backfilled to fill gaps in group metric history.

Contents

- Auto Scaling group metrics
- Dimensions for Auto Scaling group metrics
- Predictive scaling metrics and dimensions
- Enable Auto Scaling group metrics (console)
- Enable Auto Scaling group metrics (AWS CLI)

Auto Scaling group metrics

With these metrics, you get nearly continuous visibility into the history of your Auto Scaling group, such as changes in the size of the group over time.

Metric	Description
`GroupMinSize`	The minimum size of the Auto Scaling group. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupMaxSize`	The maximum size of the Auto Scaling group. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupDesiredCapacity`	The number of instances that the Auto Scaling group attempts to maintain. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupInServiceInstances`	The number of instances that are running as part of the Auto Scaling group. This metric does not include instances that are pending or terminating. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupPendingInstances`	The number of instances that are pending. A pending instance is not yet in service. This metric does not include instances that are in service or terminating. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupStandbyInstances`	The number of instances that are in a `Standby` state. Instances in this state are still running but are not actively in service. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupTerminatingInstances`	The number of instances that are in the process of terminating. This metric does not include instances that are in service or pending.

Metric	Description
	Reporting criteria: Reported if metrics collection is enabled.
`GroupTotalInstances`	The total number of instances in the Auto Scaling group. This metric identifies the number of instances that are in service, pending, and terminating. **Reporting criteria**: Reported if metrics collection is enabled.

When you configure a mixed instances group to measure its desired capacity in different units, such as by assigning weights based on the vCPU count of each instance type, the following metrics count the number of units used by your Auto Scaling group. If you did not configure a mixed instances group to measure its desired capacity in different units, then the following metrics are populated, but are equal to the metrics that are defined in the previous table. For more information, see Setup overview for creating a mixed instances group.

Metric	Description
`GroupInServiceCapacity`	The number of capacity units that are running as part of the Auto Scaling group. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupPendingCapacity`	The number of capacity units that are pending. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupStandbyCapacity`	The number of capacity units that are in a `Standby` state. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupTerminatingCapacity`	The number of capacity units that are in the process of terminating. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupTotalCapacity`	The total number of capacity units in the Auto Scaling group.

Metric	Description
	Reporting criteria: Reported if metrics collection is enabled.

Amazon EC2 Auto Scaling also reports the following metrics for Auto Scaling groups that have a warm pool. For more information, see Decrease latency for applications with long boot times using warm pools.

Metric	Description
`WarmPoolMinSize`	The minimum size of the warm pool. **Reporting criteria**: Reported if metrics collection is enabled.
`WarmPoolDesiredCapacity`	The amount of capacity that Amazon EC2 Auto Scaling attempts to maintain in the warm pool. This is equivalent to the maximum size of the Auto Scaling group minus its desired capacity, or, if set, as the maximum prepared capacity of the Auto Scaling group minus its desired capacity. However, when the minimum size of the warm pool is equal to or greater than the difference between the maximum size (or, if set, the maximum prepared capacity) and the desired capacity of the Auto Scaling group, then the warm pool desired capacity will be equivalent to the `WarmPoolMinSize` . **Reporting criteria**: Reported if metrics collection is enabled.
`WarmPoolPendingCapacity`	The amount of capacity in the warm pool that is pending. This metric does not include instances that are running, stopped, or terminating. **Reporting criteria**: Reported if metrics collection is enabled.
`WarmPoolTerminatingCapacity`	The amount of capacity in the warm pool that is in the process of terminating. This metric does not include instances that are running, stopped, or pending.

Metric	Description
	Reporting criteria: Reported if metrics collection is enabled.
`WarmPoolWarmedCapacity`	The amount of capacity available to enter the Auto Scaling group during scale out. This metric does not include instances that are pending or terminating. **Reporting criteria**: Reported if metrics collection is enabled.
`WarmPoolTotalCapacity`	The total capacity of the warm pool, including instances that are running, stopped, pending, or terminating. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupAndWarmPoolDesiredCapacity`	The desired capacity of the Auto Scaling group and the warm pool combined. **Reporting criteria**: Reported if metrics collection is enabled.
`GroupAndWarmPoolTotalCapacity`	The total capacity of the Auto Scaling group and the warm pool combined. This includes instances that are running, stopped, pending, terminating, or in service. **Reporting criteria**: Reported if metrics collection is enabled.

Dimensions for Auto Scaling group metrics

You can use the following dimensions to refine the metrics listed in the previous tables.

Dimension	Description
`AutoScalingGroupName`	Filters on the name of an Auto Scaling group.

Predictive scaling metrics and dimensions

The `AWS/AutoScaling` namespace includes the following metrics for predictive scaling.

Metrics are available with a resolution of one hour.

You can evaluate forecast accuracy by comparing forecasted values with actual values. For more information about evaluating forecast accuracy using these metrics, see Monitor predictive scaling metrics with CloudWatch.

Metric	Description	Dimensions
`PredictiveScalingLoadForecast`	The amount of load that's anticipated to be generated by your application. The `Average`, `Minimum`, and `Maximum` statistics are useful, but the `Sum` statistic is not. **Reporting criteria**: Reported after the initial forecast is created.	`AutoScalingGroupName`, `PolicyName`, `PairIndex`
`PredictiveScalingCapacityForecast`	The anticipated amount of capacity needed to meet application demand. This is based on the load forecast and target utilization level at which you want to maintain your Auto Scaling instances. The `Average`, `Minimum`, and `Maximum` statistics are useful, but the `Sum` statistic is not. **Reporting criteria**: Reported after the initial forecast is created.	`AutoScalingGroupName`, `PolicyName`
`PredictiveScalingMetricPairCorrelation`	The correlation between the scaling metric and the per-instance average of the load metric. Predictive scaling assumes high correlation. Therefore, if you observe low value for this metric, it's better not to use a metric pair. The `Average`, `Minimum`, and `Maximum` statistics are useful, but the `Sum` statistic is not.	`AutoScalingGroupName`, `PolicyName`, `PairIndex`

Metric	Description	Dimensions
	Reporting criteria: Reported after the initial forecast is created.	

> **Note**
>
> The `PairIndex` dimension returns information associated with the index of the load-scaling metric pair as assigned by Amazon EC2 Auto Scaling. Currently, the only valid value is `0`.

Enable Auto Scaling group metrics (console)

To enable group metrics

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Monitoring** tab, select the **Auto Scaling group metrics collection**, **Enable** check box located at the top of the page under **Auto Scaling**.

To disable group metrics

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select your Auto Scaling group.
3. On the **Monitoring** tab, clear the **Auto Scaling group metrics collection**, **Enable** check box.

Enable Auto Scaling group metrics (AWS CLI)

To enable Auto Scaling group metrics

Enable one or more group metrics by using the enable-metrics-collection command. For example, the following command enables a single metric for the specified Auto Scaling group.

```
aws autoscaling enable-metrics-collection --auto-scaling-group-name my-asg \
  --metrics GroupDesiredCapacity --granularity "1Minute"
```

If you omit the `--metrics` option, all metrics are enabled.

```
aws autoscaling enable-metrics-collection --auto-scaling-group-name my-asg \
  --granularity "1Minute"
```

To disable Auto Scaling group metrics

Use the disable-metrics-collection command to disable all group metrics.

```
aws autoscaling disable-metrics-collection --auto-scaling-group-name my-asg
```

Configure monitoring for Auto Scaling instances

Amazon EC2 collects and processes raw data from instances into readable, near real-time metrics that describe the CPU and other usage data for your Auto Scaling group. You can configure the interval for monitoring these metrics by choosing one-minute or five-minute granularity.

Instance monitoring is enabled whenever an instance is launched, using either basic monitoring (five-minute granularity) or detailed monitoring (one-minute granularity). For detailed monitoring, additional charges apply. For more information, see Amazon CloudWatch pricing and Monitoring your instances using CloudWatch in the *Amazon EC2 User Guide*.

Before creating an Auto Scaling group, you should create a launch template or launch configuration that permits the type of monitoring that is appropriate to your application. If you add a scaling policy to your group, we strongly recommend that you use detailed monitoring to get metric data for EC2 instances at a one-minute granularity, because that achieves a faster response to changes in load.

Contents

- Enable detailed monitoring (console)
- Enable detailed monitoring (AWS CLI)
- Switch between basic and detailed monitoring
- Collect additional metrics using the CloudWatch agent

Enable detailed monitoring (console)

By default, basic monitoring is enabled when you use the AWS Management Console to create a launch template or launch configuration.

To enable detailed monitoring in a launch template

When you create the launch template using the AWS Management Console, in the **Advanced details** section, for **Detailed CloudWatch monitoring**, choose **Enable**. Otherwise, basic monitoring is enabled. For more information, see Create a launch template using advanced settings.

To enable detailed monitoring in a launch configuration

When you create the launch configuration using the AWS Management Console, in the **Additional configuration** section, select **Enable EC2 instance detailed monitoring within CloudWatch**. Otherwise, basic monitoring is enabled. For more information, see Create a launch configuration.

Enable detailed monitoring (AWS CLI)

By default, basic monitoring is enabled when you create a launch template using the AWS CLI. Detailed monitoring is enabled by default when you create a launch configuration using the AWS CLI.

To enable detailed monitoring in a launch template

For launch templates, use the create-launch-template command and pass a JSON file that contains the information for creating the launch template. Set the monitoring attribute to `"Monitoring":{"Enabled":true}` to enable detailed monitoring or `"Monitoring":{"Enabled":false}` to enable basic monitoring.

To enable detailed monitoring in a launch configuration

For launch configurations, use the create-launch-configuration command with the `--instance-monitoring` option. Set this option to `true` to enable detailed monitoring or `false` to enable basic monitoring.

```
--instance-monitoring Enabled=true
```

Switch between basic and detailed monitoring

To change the type of monitoring enabled on new EC2 instances, update the launch template or update the Auto Scaling group to use a new launch template or launch configuration. Existing

instances continue to use the previously enabled monitoring type. To update all instances, terminate them so that they are replaced by your Auto Scaling group or update instances individually using monitor-instances and unmonitor-instances.

> **Note**
>
> With the instance refresh and maximum instance lifetime features, you can also replace all instances in the Auto Scaling group to launch new instances that use the new settings. For more information, see Replace the instances in your Auto Scaling group.

When you switch between basic and detailed monitoring:

If you have CloudWatch alarms associated with the step scaling policies or simple scaling policies for your Auto Scaling group, use the put-metric-alarm command to update each alarm. Make each period match the monitoring type (300 seconds for basic monitoring and 60 seconds for detailed monitoring). If you change from detailed monitoring to basic monitoring but do not update your alarms to match the five-minute period, they continue to check for statistics every minute. They might find no data available for as many as four out of every five periods.

Collect additional metrics using the CloudWatch agent

To collect operating system-level metrics like available and used memory, you must install the CloudWatch agent. Additional fees may apply. You can use the CloudWatch agent to collect both system metrics and log files from Amazon EC2 instances. For more information, see Metrics collected by the CloudWatch agent in the *Amazon CloudWatch User Guide*.

Log Amazon EC2 Auto Scaling API calls with AWS CloudTrail

Amazon EC2 Auto Scaling is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls for Auto Scaling as events. The calls captured include calls from the AWS Management Console and code calls to the Auto Scaling API operations. Using the information collected by CloudTrail, you can determine the request that was made to Auto Scaling, the IP address from which the request was made, when it was made, and additional details.

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

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

Auto Scaling management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

Amazon EC2 Auto Scaling logs all Auto Scaling control plane operations as management events. For a list of the Amazon EC2 Auto Scaling control plane operations that Auto Scaling logs to CloudTrail, see the Amazon EC2 Auto Scaling API Reference.

Auto Scaling event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail event that demonstrates the `CreateLaunchConfiguration` operation.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-21T17:05:42Z"
            }
        }
    },
    "eventTime": "2018-08-21T17:07:49Z",
    "eventSource": "autoscaling.amazonaws.com",
    "eventName": "CreateLaunchConfiguration",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Coral/Jakarta",
    "requestParameters": {
        "ebsOptimized": false,
        "instanceMonitoring": {
            "enabled": false
        },
        "instanceType": "t2.micro",
        "keyName": "EC2-key-pair-oregon",
        "blockDeviceMappings": [
            {
                "deviceName": "/dev/xvda",
                "ebs": {
                    "deleteOnTermination": true,
```

```
                    "volumeSize": 8,
                    "snapshotId": "snap-01676e0a2c3c7de9e",
                    "volumeType": "gp2"
                }
            }
        ],
        "launchConfigurationName": "launch_configuration_1",
        "imageId": "ami-6cd6f714d79675a5",
        "securityGroups": [
            "sg-00c429965fd921483"
        ]
    },
    "responseElements": null,
    "requestID": "0737e2ea-fb2d-11e3-bfd8-99133058e7bb",
    "eventID": "3fcfb182-98f8-4744-bd45-b38835ab61cb",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Amazon SNS notification options for Amazon EC2 Auto Scaling

You can configure your Auto Scaling group to notify you of important events that affect your application. With notifications, you can also eliminate polling, and you won't encounter the `RequestLimitExceeded` error that sometimes results from polling.

There are two ways to receive notifications about Amazon EC2 Auto Scaling:

- **Amazon Simple Notification Service** – Amazon SNS can notify you when your Auto Scaling group launches or terminates instances. You can only turn Amazon SNS notifications on or off. For more information, see Amazon SNS and Amazon EC2 Auto Scaling.
- **Amazon EventBridge** – EventBridge provides more advanced, event-driven notifications matched to specified criteria and sent to a variety of targets, including Amazon SNS. EventBridge can also monitor a wider range of Auto Scaling events for more precise monitoring. For more information, see Use EventBridge to handle Auto Scaling events.

You can optionally use notifications with lifecycle hooks to perform custom actions on instances during launch or termination. For more information on how to configure the notifications to use with lifecycle hooks, see Amazon EC2 Auto Scaling lifecycle hooks.

Amazon SNS and Amazon EC2 Auto Scaling

This section shows how to use Amazon SNS to monitor when your Auto Scaling group launches or terminates instances.

For example, if you configure your Auto Scaling group to use the `autoscaling:EC2_INSTANCE_TERMINATE` notification type, and your Auto Scaling group terminates an instance, it sends an email notification. This email contains the details of the terminated instance, such as the instance ID and the reason that the instance was terminated.

Note that as Amazon EC2 Auto Scaling adds or removes instances from the group, notifications about these changes are sent to you, with one notification sent per instance. However, delivery of these notifications is on a best-effort basis, and your instances could still fail after the initial notification, for example, if a later health check fails. For more information about the health check process, see Health checks for instances in an Auto Scaling group.

For more information about Amazon SNS generally, see the Amazon Simple Notification Service Developer Guide.

Contents

- SNS notifications
- Configure Amazon SNS notifications for Amazon EC2 Auto Scaling
  - Create an Amazon SNS topic
  - Subscribe to the Amazon SNS topic
  - Confirm your Amazon SNS subscription
  - Configure your Auto Scaling group to send notifications
  - Test the notification
  - Delete the notification configuration
- Key policy for an encrypted Amazon SNS topic

SNS notifications

Amazon EC2 Auto Scaling supports sending Amazon SNS notifications when the following events occur.

Event	Description
`autoscaling:EC2_INSTANCE_LAUNCH`	Successful instance launch
`autoscaling:EC2_INSTANCE_LAUNCH_ERROR`	Failed instance launch
`autoscaling:EC2_INSTANCE_TERMINATE`	Successful instance termination
`autoscaling:EC2_INSTANCE_TERMINATE_ERROR`	Failed instance termination

The message includes the following information:

- `Event` — The event.
- `AccountId` — The Amazon Web Services account ID.
- `AutoScalingGroupName` — The name of the Auto Scaling group.
- `AutoScalingGroupARN` — The ARN of the Auto Scaling group.
- `EC2InstanceId` — The ID of the EC2 instance.

For example:

```
Service: AWS Auto Scaling
Time: 2016-09-30T19:00:36.414Z
RequestId: 4e6156f4-a9e2-4bda-a7fd-33f2ae528958
Event: autoscaling:EC2_INSTANCE_LAUNCH
AccountId: 123456789012
AutoScalingGroupName: my-asg
AutoScalingGroupARN: arn:aws:autoscaling:region:123456789012:autoScalingGroup...
ActivityId: 4e6156f4-a9e2-4bda-a7fd-33f2ae528958
Description: Launching a new EC2 instance: i-0598c7d356eba48d7
```

```
Cause: At 2016-09-30T18:59:38Z a user request update of AutoScalingGroup constraints
 to ...
StartTime: 2016-09-30T19:00:04.445Z
EndTime: 2016-09-30T19:00:36.414Z
StatusCode: InProgress
StatusMessage:
Progress: 50
EC2InstanceId: i-0598c7d356eba48d7
Details: {"Subnet ID":"subnet-id","Availability Zone":"zone"}
Origin: AutoScalingGroup
Destination: EC2
```

Configure Amazon SNS notifications for Amazon EC2 Auto Scaling

To use Amazon SNS to send email notifications, you must first create a *topic* and then subscribe your email addresses to the topic.

Create an Amazon SNS topic

An SNS topic is a logical access point, a communication channel your Auto Scaling group uses to send the notifications. You create a topic by specifying a name for your topic.

When you create a topic name, the name must meet the following requirements:

- Between 1 and 256 characters long
- Contain uppercase and lowercase ASCII letters, numbers, underscores, or hyphens

For more information, see Creating an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

Subscribe to the Amazon SNS topic

To receive the notifications that your Auto Scaling group sends to the topic, you must subscribe an endpoint to the topic. In this procedure, for **Endpoint**, specify the email address where you want to receive the notifications from Amazon EC2 Auto Scaling.

For more information, see Subscribing to an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

Confirm your Amazon SNS subscription

Amazon SNS sends a confirmation email to the email address you specified in the previous step.

Make sure that you open the email from AWS Notifications and choose the link to confirm the subscription before you continue with the next step.

You will receive an acknowledgment message from AWS. Amazon SNS is now configured to receive notifications and send the notification as an email to the email address that you specified.

Configure your Auto Scaling group to send notifications

You can configure your Auto Scaling group to send notifications to Amazon SNS when a scaling event, such as launching instances or terminating instances, takes place. Amazon SNS sends a notification with information about the instances to the email address that you specified.

To configure Amazon SNS notifications for your Auto Scaling group (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom part of the page, showing information about the group that's selected.
3. On the **Activity** tab, choose **Activity notifications**, **Create notification**.
4. On the **Create notifications** pane, do the following:

   a. For **SNS Topic**, select your SNS topic.

   b. For **Event types**, select the events to send the notifications.

   c. Choose **Create**.

To configure Amazon SNS notifications for your Auto Scaling group (AWS CLI)

Use the following put-notification-configuration command.

```
aws autoscaling put-notification-configuration --auto-scaling-group-name my-
asg --topic-arn arn --notification-types "autoscaling:EC2_INSTANCE_LAUNCH"
 "autoscaling:EC2_INSTANCE_TERMINATE"
```

Test the notification

To generate a notification for a launch event, update the Auto Scaling group by increasing the desired capacity of the Auto Scaling group by 1. You receive a notification within a few minutes after instance launch.

To change the desired capacity (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom part of the **Auto Scaling groups** page, showing information about the group that's selected.
3. On the **Details** tab, choose **Group details**, **Edit**.
4. For **Desired capacity**, increase the current value by 1. If this value exceeds **Maximum capacity**, you must also increase the value of **Maximum capacity** by 1.
5. Choose **Update**.
6. After a few minutes, you'll receive notification for the event. If you do not need the additional instance that you launched for this test, you can decrease **Desired capacity** by 1. After a few minutes, you'll receive notification for the event.

Delete the notification configuration

You can delete your Amazon EC2 Auto Scaling notification configuration if it is no longer being used.

To delete Amazon EC2 Auto Scaling notification configuration (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select your Auto Scaling group.
3. On the **Activity** tab, select the check box next to the notification you want to delete and then choose **Actions**, **Delete**.

To delete Amazon EC2 Auto Scaling notification configuration (AWS CLI)

Use the following **delete-notification-configuration** command.

```
aws autoscaling delete-notification-configuration --auto-scaling-group-name my-asg --
topic-arn arn
```

For information about deleting the Amazon SNS topic and all subscriptions associated with your Auto Scaling group, see Deleting an Amazon SNS subscription and topic in the *Amazon Simple Notification Service Developer Guide*.

Key policy for an encrypted Amazon SNS topic

The Amazon SNS topic you specify might be encrypted with a customer managed key created with the AWS Key Management Service. To give Amazon EC2 Auto Scaling permission to publish to encrypted topics, you must first create your KMS key and then add the following statement to the policy of the KMS key. Replace the example ARN with the ARN of the appropriate service-linked role that is allowed access to the key. For more information, see Configuring AWS KMS permissions in the *Amazon Simple Notification Service Developer Guide*.

In this example, the policy statement gives the service-linked role named **AWSServiceRoleForAutoScaling** permissions to use the customer managed key. To learn more about the Amazon EC2 Auto Scaling service-linked role, see Service-linked roles for Amazon EC2 Auto Scaling.

```
{
  "Sid": "Allow service-linked role use of the customer managed key",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::123456789012:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling"
  },
  "Action": [
    "kms:GenerateDataKey*",
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

The `aws:SourceArn` and `aws:SourceAccount` condition keys are not supported in key policies that allow Amazon EC2 Auto Scaling to publish to encrypted topics.

AWS services integrated with Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling can be integrated with other AWS services. Review the following integration options to learn more about how each service works with Amazon EC2 Auto Scaling.

Topics

- Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions
- Reserve capacity in specific Availability Zones with Capacity Reservations
- Create Auto Scaling groups from the command line using AWS CloudShell
- Create Auto Scaling groups with AWS CloudFormation
- Get instance type recommendations with AWS Compute Optimizer
- Use Elastic Load Balancing; to distribute incoming application traffic in your Auto Scaling group
- Manage traffic flow with a VPC Lattice target group
- Use EventBridge to handle Auto Scaling events
- Provide network connectivity for your Auto Scaling instances using Amazon VPC

Use Capacity Rebalancing to handle Amazon EC2 Spot interruptions

You can configure Amazon EC2 Auto Scaling to monitor and automatically respond to changes that affect the availability of your Spot Instances. Capacity Rebalancing helps you maintain workload availability by proactively augmenting your fleet with a new Spot Instance before a running instance is interrupted by Amazon EC2.

The goal of Capacity Rebalancing is to keep processing your workload without interruption. When Spot Instances are at an elevated risk of interruption, the Amazon EC2 Spot service notifies Amazon EC2 Auto Scaling with an EC2 instance rebalance recommendation.

When you enable Capacity Rebalancing for your Auto Scaling group, Amazon EC2 Auto Scaling attempts to proactively replace the Spot Instances in your group that have received a rebalance recommendation. This provides an opportunity to rebalance your workload to new Spot Instances that aren't at an elevated risk of interruption. Your workload can continue to process the work while Amazon EC2 Auto Scaling launches new Spot Instances before your existing instances are interrupted.

When you don't use Capacity Rebalancing, Amazon EC2 Auto Scaling doesn't replace Spot Instances until after the Amazon EC2 Spot service interrupts the instances and their health check fails. Before interrupting an instance, Amazon EC2 always gives both an EC2 instance rebalance recommendation and a Spot two-minute instance interruption notice.

Contents

- Overview
- Capacity Rebalancing behavior
- Considerations
- Enable Capacity Rebalancing using the AWS Management Console or AWS CLI

Overview

To use Capacity Rebalancing with your Auto Scaling group, the basic steps are:

1. Configure your Auto Scaling group to use multiple instance types and Availability Zones. This way, Amazon EC2 Auto Scaling can look at the available capacity for Spot Instances in each Availability Zone. For more information, see Auto Scaling groups with multiple instance types and purchase options.
2. Add lifecycle hooks as needed to perform a graceful shutdown of your application inside the instances that receive the rebalance notification. For more information, see Amazon EC2 Auto Scaling lifecycle hooks.

   The following are some reasons why you might use a lifecycle hook:

   - For graceful shutdown of Amazon SQS workers
   - To complete deregistration from the Domain Name System (DNS)
   - To pull system or application logs and upload them to Amazon Simple Storage Service (Amazon S3)
3. Develop a custom action for the lifecycle hook. To invoke your custom action as soon as possible, you need to know when an instance is ready to be terminated. Find this out by detecting the lifecycle state of the instance.
   - To invoke an action outside of the instance, write an EventBridge rule and automate what action to take when an event pattern matches the rule.
   - To invoke an action inside of the instance, configure the instance to run a shutdown script and retrieve the lifecycle state through instance metadata.

It's critical to design the custom action to finish in under two minutes. This makes sure there's enough time to complete tasks before instance termination.

After you complete these steps, you can begin using Capacity Rebalancing.

Capacity Rebalancing behavior

With Capacity Rebalancing, Amazon EC2 Auto Scaling behaves in the following way when an instance receives a rebalance recommendation:

- When the new Spot Instance launches, Amazon EC2 Auto Scaling waits until the new instance passes its health check before it terminates the previous instance. When replacing more than one instance, the termination of each previous instance starts after the new instance has launched and passed its health check.
- Because Amazon EC2 Auto Scaling attempts to launch new instances before terminating previous ones, being at or near the specified maximum capacity could impede or completely halt rebalancing activities. To avoid this problem, Amazon EC2 Auto Scaling can temporarily exceed the group's maximum size by up to 10 percent of the desired capacity.
- If you didn't add a lifecycle hook to your Auto Scaling group, Amazon EC2 Auto Scaling starts terminating the previous instances as soon as the new instances pass their health check.
- If you added a lifecycle hook, this extends the amount of time it takes before we start terminating the previous instances by the timeout value you specified for the lifecycle hook.
- If you are using scaling policies or scheduled scaling, the scaling activities run in parallel. If a scaling activity is in progress and your Auto Scaling group is below its new desired capacity, Amazon EC2 Auto Scaling scales out first before terminating the previous instances.

If there is no capacity for your instance types in one Availability Zone, Amazon EC2 Auto Scaling keeps trying to launch Spot Instances in other enabled Availability Zones until it succeeds.

In the worst case scenario, if new instances fail to launch or their health checks fail, Amazon EC2 Auto Scaling keeps trying to relaunch them. While it's trying to launch new instances, your previous ones will eventually be interrupted and forcibly terminated with a two-minute interruption notice.

Considerations

Consider the following when using Capacity Rebalancing:

Design your application to be tolerant to Spot interruptions

Your application should be able to handle dynamic changes in the number of instances and the possibility of a Spot Instance being interrupted early. For example, if your Auto Scaling group is behind an Elastic Load Balancing load balancer, Amazon EC2 Auto Scaling waits for the instance to deregister from the load balancer before calling your lifecycle hook. If the time to deregister the instance and complete the lifecycle action takes too long, the instance might be interrupted while Amazon EC2 Auto Scaling waits for your lifecycle action to complete before terminating the instance.

It's not always possible for Amazon EC2 to send the rebalance recommendation signal before the two-minute Spot Instance interruption notice. Sometimes, the rebalance recommendation signal arrives at the same time as the two-minute interruption notice. When this happens, Amazon EC2 Auto Scaling calls the lifecycle hook and attempts to launch a new Spot Instance immediately.

Avoid an elevated risk of interruption of replacement Spot Instances

Your replacement Spot Instances might be at an elevated risk of interruption if you use the `lowest-price` allocation strategy. This is because we launch instances in the lowest priced pool that has available capacity at that moment, even if your replacement Spot Instances are likely to be interrupted soon after they launch. To avoid an elevated risk of interruption, we strongly recommend that you do not use the `lowest-price` allocation strategy. Instead, we recommend the `price-capacity-optimized` allocation strategy. This strategy launches replacement Spot Instances in Spot pools that are least likely to be interrupted and have the lowest possible price. Therefore, they're less likely to be interrupted in the near future.

Amazon EC2 Auto Scaling will only launch a new instance if availability is the same or better

One of the goals of Capacity Rebalancing is to improve a Spot Instance's availability. If an existing Spot Instance receives a rebalance recommendation, Amazon EC2 Auto Scaling will only launch a new instance if the new instance provides the same or better availability than the existing instance. If the risk of interruption of a new instance will be worse than the existing instance, then Amazon EC2 Auto Scaling will not launch a new instance. Amazon EC2 Auto Scaling will, however, continue to assess the Spot capacity pools based on information provided by the Amazon EC2 Spot service, and will launch a new instance if availability improves.

There is a chance that your existing instance will be interrupted without Amazon EC2 Auto Scaling proactively launching a new instance. When this happens, Amazon EC2 Auto Scaling

attempts to launch a new instance as soon as it receives the Spot Instance interruption notice. This happens regardless of whether the new instance has a high risk of interruption.

Capacity Rebalancing does not increase your Spot Instance interruption rate

When you enable Capacity Rebalancing, it does not increase your Spot Instance interruption rate (the number of Spot Instances that are reclaimed when Amazon EC2 needs the capacity back). However, if Capacity Rebalancing detects an instance is at risk of interruption, Amazon EC2 Auto Scaling will immediately attempt to launch a new instance. Therefore, more instances might be replaced than if you waited for Amazon EC2 Auto Scaling to launch a new instance after the at-risk instance was interrupted.

While you might replace more instances with Capacity Rebalancing enabled, you benefit from being proactive rather than reactive. This gives you more time to take action before your instances are interrupted. With a Spot Instance interruption notice, you typically only have up to two minutes to gracefully shut down your instance. With Capacity Rebalancing launching a new instance in advance, you give existing processes a better chance of completing on your at-risk instance. You can also start your instance shutdown procedures, prevent new work from being scheduled on your at-risk instance, and prepare the newly launched instance to take over the application. With proactive replacement in Capacity Rebalancing, you benefit from graceful continuity.

The following theoretical example demonstrates the risks and benefits of using Capacity Rebalancing:

- 2:00 PM – A rebalance recommendation is received for instance A. Amazon EC2 Auto Scaling immediately attempts to launch replacement instance B, giving you time to start your shutdown procedures.
- 2:30 PM – A rebalance recommendation is received for instance B, which is replaced with instance C. This gives you time to start your shutdown procedures.
- 2:32 PM – If Capacity Rebalancing isn’t enabled, and if a Spot Instance interruption notice would've been received at 2:32 PM for instance A, you would have had only two minutes to take action. However, Instance A would have continued running until this time.

Enable Capacity Rebalancing using the AWS Management Console or AWS CLI

You can use the AWS Management Console or AWS CLI to enable Capacity Rebalancing for your Auto Scaling group. Amazon EC2 Auto Scaling attempts to proactively replace the Spot Instances in your group that have received a rebalance recommendation.

Enable Capacity Rebalancing (console)

You can enable or disable Capacity Rebalancing when you create or update an Auto Scaling group.

To enable Capacity Rebalancing for a new Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Choose **Create Auto Scaling group**.
3. For **Step 1: Choose launch template or configuration**, enter a name for the Auto Scaling group, choose a launch template, and then choose **Next** to proceed to the next step.
4. For **Step 2: Choose instance launch options**, for **Instance type requirements**, choose settings to create a mixed instances group. This includes the instance types that it can launch, instance purchase options, and allocation strategies for Spot and On-Demand Instances. By default, these settings are not configured. To configure them, you must select **Override launch template**. For more information about creating a mixed instances group, see Auto Scaling groups with multiple instance types and purchase options.
5. Under **Network**, choose the options as desired. Verify that the subnets you want to use are in different Availability Zones.
6. Under the **Allocation strategies** section, choose a Spot allocation strategy. Enable or disable Capacity Rebalancing by selecting or clearing the check box under **Capacity Rebalancing**. You only see this option when you request a percentage of the Auto Scaling group to be launched as Spot Instances in the **Instance purchase options** section.
7. Create the Auto Scaling group.
8. (Optional) Add lifecycle hooks as needed. For more information, see Add lifecycle hooks to your Auto Scaling group.

To enable or disable Capacity Rebalancing for an existing Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group. A split pane opens in the bottom of the page.
3. On the **Details** tab, choose **Allocation strategies**, **Edit**.
4. Under the **Allocation strategies** section, enable or disable Capacity Rebalancing by selecting or clearing the check box under **Capacity Rebalancing**.
5. Choose **Update**.

Enable Capacity Rebalancing (AWS CLI)

The following examples show how to use the AWS CLI to enable and disable Capacity Rebalancing.

Use the create-auto-scaling-group or update-auto-scaling-group command with the following parameter:

- `--capacity-rebalance` / `--no-capacity-rebalance` – Boolean value that indicates whether Capacity Rebalancing is enabled.

Before you call the create-auto-scaling-group command, you need the name of a launch template that is configured for use with an Auto Scaling group. For more information, see Create a launch template for an Auto Scaling group.

> **Note**
>
> The following procedures show how to use a configuration file formatted in JSON or YAML. If you use AWS CLI version 1, you must specify a JSON-formatted configuration file. If you use AWS CLI version 2, you can specify a configuration file formatted in either YAML or JSON.

JSON

To create and configure a new Auto Scaling group

- Use the following create-auto-scaling-group command to create a new Auto Scaling group and enable Capacity Rebalancing. This command references a JSON file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-json file://~/config.json
```

If you don't already have a CLI configuration file that specifies a mixed instances policy, create one.

Add the following line to the top-level JSON object in the configuration file.

```
{
    "CapacityRebalance": true
}
```

The following is an example `config.json` file.

```
{
    "AutoScalingGroupName": "my-asg",
    "DesiredCapacity": 12,
    "MinSize": 12,
    "MaxSize": 15,
    "CapacityRebalance": true,
    "MixedInstancesPolicy": {
        "InstancesDistribution": {
            "OnDemandBaseCapacity": 0,
            "OnDemandPercentageAboveBaseCapacity": 25,
            "SpotAllocationStrategy": "price-capacity-optimized"
        },
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "$Default"
            },
            "Overrides": [
                {
                    "InstanceType": "c5.large"
```

```
                },
                {
                    "InstanceType": "c5a.large"
                },
                {
                    "InstanceType": "m5.large"
                },
                {
                    "InstanceType": "m5a.large"
                },
                {
                    "InstanceType": "c4.large"
                },
                {
                    "InstanceType": "m4.large"
                },
                {
                    "InstanceType": "c3.large"
                },
                {
                    "InstanceType": "m3.large"
                }
            ]
        }
    },
    "TargetGroupARNs": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/my-alb-target-group/943f017f100becff",
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
}
```

YAML

To create and configure a new Auto Scaling group

- Use the following create-auto-scaling-group command to create a new Auto Scaling group and enable Capacity Rebalancing. This command references a YAML file as the sole parameter for your Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --cli-input-yaml file://~/config.yaml
```

Add the following line to your configuration file formatted in YAML.

```
CapacityRebalance: true
```

The following is an example `config.yaml` file.

```
---
AutoScalingGroupName: my-asg
DesiredCapacity: 12
MinSize: 12
MaxSize: 15
CapacityRebalance: true
MixedInstancesPolicy:
  InstancesDistribution:
    OnDemandBaseCapacity: 0
    OnDemandPercentageAboveBaseCapacity: 25
    SpotAllocationStrategy: price-capacity-optimized
  LaunchTemplate:
    LaunchTemplateSpecification:
      LaunchTemplateName: my-launch-template
      Version: $Default
    Overrides:
    - InstanceType: c5.large
    - InstanceType: c5a.large
    - InstanceType: m5.large
    - InstanceType: m5a.large
    - InstanceType: c4.large
    - InstanceType: m4.large
    - InstanceType: c3.large
    - InstanceType: m3.large
TargetGroupARNs:
- arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/my-alb-target-
group/943f017f100becff
VPCZoneIdentifier: subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782
```

To enable Capacity Rebalancing for an existing Auto Scaling group

- Use the following update-auto-scaling-group command to enable Capacity Rebalancing.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --capacity-rebalance
```

To verify that Capacity Rebalancing is enabled for an Auto Scaling group

- Use the following describe-auto-scaling-groups command to verify that Capacity Rebalancing is enabled and to view the details.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
```

The following is an example response.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg",
            "AutoScalingGroupARN": "arn",
            ...
            "CapacityRebalance": true
        }
    ]
}
```

To disable Capacity Rebalancing

Use the update-auto-scaling-group command with the `--no-capacity-rebalance` option to disable Capacity Rebalancing.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --no-capacity-rebalance
```

Related resources

For more information about Capacity Rebalancing, see Proactively manage Spot Instance lifecycle using the new Capacity Rebalancing feature for EC2 Auto Scaling on the AWS Compute Blog.

For more information about the EC2 instance rebalance recommendations, see EC2 instance rebalance recommendations in the *Amazon EC2 User Guide*.

To learn more about lifecycle hooks, see the following resources.

- Tutorial: Configure a lifecycle hook that invokes a Lambda function (using EventBridge)
- Tutorial: Use data script and instance metadata to retrieve lifecycle state

Limitations

- Amazon EC2 Auto Scaling can replace the instance that receives the rebalance notification only if the instance isn't protected from scale in. However, scale-in protection doesn't prevent termination from a Spot interruption. For more information, see Use instance scale-in protection to control instance termination.
- Support for Capacity Rebalancing is available in all commercial AWS Regions where Amazon EC2 Auto Scaling is available, except for the Middle East (UAE) Region.

Reserve capacity in specific Availability Zones with Capacity Reservations

Amazon EC2 On-Demand Capacity Reservations help you reserve compute capacity in specific Availability Zones. To start using Capacity Reservations, you create a Capacity Reservation in a specific Availability Zone. Then, you can launch instances into the reserved capacity, view its capacity utilization in real time, and increase or decrease its capacity as needed.

Capacity Reservations are configured as either `open` or `targeted`. If the Capacity Reservation is `open`, all new and existing instances that have matching attributes automatically run in the capacity of the Capacity Reservation. If the Capacity Reservation is `targeted`, instances must specifically target it to run in the reserved capacity.

This topic shows how to create an Auto Scaling group that launches On-Demand Instances into `targeted` Capacity Reservations. This gives you more control over when to use specific Capacity Reservations.

The basic steps are:

1. Create Capacity Reservations in multiple Availability Zones that have the same instance type, platform, and instance count.
2. Group Capacity Reservations using AWS Resource Groups.
3. Create an Auto Scaling group with a launch template that targets the resource group, using the same Availability Zones as the Capacity Reservations.

Contents

- Step 1: Create the Capacity Reservations

- Step 2: Create a Capacity Reservation group
- Step 3: Create a launch template
- Step 4: Create an Auto Scaling group
- Related resources

Step 1: Create the Capacity Reservations

The first step is to create a Capacity Reservation in each Availability Zone where your Auto Scaling group will be deployed.

> **Note**
>
> You can only create `targeted` reservations when you first create the Capacity Reservations.

Console

To create your Capacity Reservations

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Capacity Reservations**, and then choose **Create Capacity Reservation**.
3. On the **Create a Capacity Reservation** page, pay attention to following settings in the **Instance details** section. The instance type, platform, and Availability Zone of the instances that you launch must match the instance type, platform, and Availability Zone that you specify here or the Capacity Reservation is not applied.

   a. For **Instance type**, choose the type of instance to launch into the reserved capacity.

   b. For **Platform**, choose the operating system for your instances.

   c. For **Availability Zone**, choose the first Availability Zone that you want to reserve capacity in.

   d. For **Total capacity**, choose the number of instances you need. Calculate the total number of instances you need for your Auto Scaling group divided by the number of Availability Zones you plan to use.
4. Under **Capacity Reservation details**, for **Capacity Reservation ends**, choose one of the following options:

- **At specific time** – Cancel the Capacity Reservation automatically at the specified date and time.
- **Manually** – Reserve the capacity until you explicitly cancel it.

5. For **Instance eligibility**, choose **Targeted: Only instances that target the Capacity Reservation**.
6. (Optional) For **Tags**, specify any tags to associate with the Capacity Reservation.
7. Choose **Create**.
8. Note the ID of the newly created Capacity Reservation. You need it to set up the Capacity Reservation group.

Repeat this procedure for each Availability Zone you want to enable for your Auto Scaling group, changing only the value of the **Availability Zone** option.

AWS CLI

To create your Capacity Reservations

Use the following create-capacity-reservation command to create the Capacity Reservations. Replace the sample values for `--availability-zone`, `--instance-type`, `--instance-platform`, and `--instance-count`.

```
aws ec2 create-capacity-reservation \
    --availability-zone us-east-1a \
    --instance-type c5.xlarge \
    --instance-platform Linux/UNIX \
    --instance-count 3 \
    --instance-match-criteria targeted
```

Example of resulting Capacity Reservation ID

```
{
    "CapacityReservation": {
        "CapacityReservationId": "cr-1234567890abcdef1",
        "OwnerId": "123456789012",
        "CapacityReservationArn": "arn:aws:ec2:us-east-1:123456789012:capacity-
reservation/cr-1234567890abcdef1",
        "InstanceType": "c5.xlarge",
        "InstancePlatform": "Linux/UNIX",
        "AvailabilityZone": "us-east-1a",
```

```
        "Tenancy": "default",
        "TotalInstanceCount": 3,
        "AvailableInstanceCount": 3,
        "EbsOptimized": false,
        "EphemeralStorage": false,
        "State": "active",
        "StartDate": "2023-07-26T21:36:14+00:00",
        "EndDateType": "unlimited",
        "InstanceMatchCriteria": "targeted",
        "CreateDate": "2023-07-26T21:36:14+00:00"
    }
}
```

Note the ID of the newly created Capacity Reservation. You need it to set up the Capacity Reservation group.

Repeat this command for each Availability Zone you want to enable for your Auto Scaling group, changing only the value of the `--availability-zone` option.

Step 2: Create a Capacity Reservation group

When you finish creating the Capacity Reservations, you can group them together using the AWS Resource Groups service. AWS Resource Groups supports several different types of groups for different uses. Amazon EC2 uses a special-purpose group, known as a service-linked resource group, to target a group of Capacity Reservations. To interact with this service-linked resource group, you can use the AWS CLI or an SDK but not the console. For more information about service-linked resource groups, see Service configurations for resource groups in the *AWS Resource Groups User Guide*.

To create a Capacity Reservation group using the AWS CLI

Use the create-group command to create a resource group that can contain only Capacity Reservations. In this example, the resource group is named *my-cr-group*.

```
aws resource-groups create-group \
    --name my-cr-group \
    --configuration '{"Type":"AWS::EC2::CapacityReservationPool"}'
 '{"Type":"AWS::ResourceGroups::Generic", "Parameters": [{"Name": "allowed-resource-
types", "Values": ["AWS::EC2::CapacityReservation"]}]}'
```

The following is an example response.

```
{
    "Group": {
        "GroupArn": "arn:aws:resource-groups:us-east-1:123456789012:group/my-cr-group",
        "Name": "my-cr-group"
    },
    "GroupConfiguration": {
        "Configuration": [
            {
                "Type": "AWS::EC2::CapacityReservationPool"
            },
            {
                "Type": "AWS::ResourceGroups::Generic",
                "Parameters": [
                    {
                        "Name": "allowed-resource-types",
                        "Values": [
                            "AWS::EC2::CapacityReservation"
                        ]
                    }
                ]
            }
        ],
        "Status": "UPDATE_COMPLETE"
    }
}
```

Note the ARN of the resource group. You need it to set up the launch template for your Auto Scaling group.

To associate your Capacity Reservations to the newly created group using the AWS CLI

Use the following group-resources command to associate the Capacity Reservations to the newly created Capacity Reservation group. For the `--resource-arns` option, specify the Capacity Reservations using their ARNs. Construct the ARNs using the relevant Region, your account ID, and the reservation IDs you noted earlier. In this example, the reservations with IDs *cr-1234567890abcdef1* and *cr-54321abcdef567890* will be grouped together in the group named *my-cr-group*.

```
aws resource-groups group-resources \
    --group my-cr-group \
    --resource-arns \
      arn:aws:ec2:region:account-id:capacity-reservation/cr-1234567890abcdef1 \
```

```
        arn:aws:ec2:region:account-id:capacity-reservation/cr-54321abcdef567890
```

The following is an example response.

```
{
   "Succeeded": [
   "arn:aws:ec2:us-east-1:123456789012:capacity-reservation/cr-1234567890abcdef1",
   "arn:aws:ec2:us-east-1:123456789012:capacity-reservation/cr-54321abcdef567890"
   ],
   "Failed": [],
   "Pending": []
}
```

For information about modifying or deleting the resource group, see the AWS Resource Groups API Reference.

Step 3: Create a launch template

Console

To create a launch template

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Instances**, choose **Launch Templates**.
3. Choose **Create launch template**. Enter a name and provide a description for the initial version of the launch template.
4. Under **Auto Scaling guidance**, select the check box.
5. Create the launch template. Choose an AMI and instance type that matches the Capacity Reservations you are planning to use, and optionally, a key pair, one or more security groups, and any additional EBS volumes or instance store volumes for your instances.
6. Expand **Advanced details**, and do the following:
   a. For **Capacity Reservation**, choose **Target by group**.
   b. For **Capacity reservation - Target by group**, choose the Capacity Reservations group that you created in the previous section, and then choose **Save**.
7. Choose **Create launch template**.
8. On the confirmation page, choose **Create Auto Scaling group**.

AWS CLI

To create a launch template

Use the following create-launch-template command to create a launch template that specifies that the Capacity Reservation targets a specific resource group. Replace the sample value for `--launch-template-name`. Replace *`c5.xlarge`* with the instance type that you used in the Capacity Reservation and *`ami-0123456789EXAMPLE`* with the ID of the AMI that you want to use. Replace *`arn:aws:resource-groups:region:account-id:group/my-cr-group`* with the ARN of the resource group that you created in the beginning of the previous section.

```
aws ec2 create-launch-template \
    --launch-template-name my-launch-template \
    --launch-template-data \
        '{"InstanceType": "c5.xlarge",
          "ImageId": "ami-0123456789EXAMPLE",
          "CapacityReservationSpecification":
            {"CapacityReservationTarget":
                { "CapacityReservationResourceGroupArn": "arn:aws:resource-
groups:region:account-id:group/my-cr-group" }
            }
        }'
```

The following is an example response.

```
{
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-0dd77bd41dEXAMPLE",
        "LaunchTemplateName": "my-launch-template",
        "CreateTime": "2023-07-26T21:42:48+00:00",
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "DefaultVersionNumber": 1,
        "LatestVersionNumber": 1
    }
}
```

Step 4: Create an Auto Scaling group

Console

Create your Auto Scaling group as you usually do, but when you choose your VPC subnets, choose a subnet from each Availability Zone that matches the `targeted` Capacity Reservations you created. Then, when your Auto Scaling group launches an On-Demand Instance in one of these Availability Zones, the instance will be run in the reserved capacity for that Availability Zone. If the resource group runs out of Capacity Reservations before your desired capacity is fulfilled, we launch anything beyond the reserved capacity as regular On-Demand capacity.

To create a simple Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the same AWS Region that you used when you created the launch template.
3. Choose **Create an Auto Scaling group**.
4. On the **Choose launch template or configuration** page, for **Auto Scaling group name**, enter a name for your Auto Scaling group.
5. For **Launch template**, choose an existing launch template.
6. For **Launch template version**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.
7. On the **Choose instance launch options** page, skip the **Instance type requirements** section to use the EC2 instance type that is specified in the launch template.
8. Under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch template. If you didn't specify a security group in your launch template, you can choose any VPC that has subnets in the same Availability Zones as your Capacity Reservations.
9. For **Availability Zones and subnets**, choose subnets from each Availability Zone that you want to include, based on which Availability Zones your Capacity Reservations are in.
10. Choose **Next** twice.
11. On the **Configure group size and scaling policies** page, for **Desired capacity**, enter the initial number of instances to launch. When you change this number to a value outside of the minimum or maximum capacity limits, you must update the values of **Minimum**

capacity or **Maximum capacity**. For more information, see Set scaling limits for your Auto Scaling group.

12. Choose **Skip to review**.
13. On the **Review** page, choose **Create Auto Scaling group**.

AWS CLI

To create a simple Auto Scaling group

Use the following create-auto-scaling-group command and specify the name and version of your launch template as the value for the `--launch-template` option. Replace the sample values for `--auto-scaling-group-name`, `--min-size`, `--max-size`, and `--vpc-zone-identifier`.

For the `--availability-zones` option, specify the Availability Zones that you created Capacity Reservations for. For example, if your Capacity Reservations specify the `us-east-1a` and `us-east-1b` Availability Zones, then you must create your Auto Scaling group in the same zones. Then, when your Auto Scaling group launches an On-Demand Instance in one of these Availability Zones, the instance will be run in the reserved capacity for that Availability Zone. If the resource group runs out of Capacity Reservations before your desired capacity is fulfilled, we launch anything beyond the reserved capacity as regular On-Demand capacity.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-template LaunchTemplateName=my-launch-template,Version='1' \
    --min-size 6 \
    --max-size 6 \
    --vpc-zone-identifier "subnet-5f46ec3b,subnet-0ecac448" \
    --availability-zones us-east-1a us-east-1b
```

Related resources

For an example implementation, see the AWS CloudFormation template in the following AWS samples GitHub repository: https://github.com/aws-samples/aws-auto-scaling-backed-by-on-demand-capacity-reservations/.

The following related topics can be helpful as you learn about Capacity Reservations.

- On-Demand Capacity Reservations
  - Create a Capacity Reservation in the *Amazon EC2 User Guide*
  - On-Demand Capacity Reservations in the *Amazon EC2 User Guide*
  - Target a group of Amazon EC2 On-Demand Capacity Reservations on the AWS Cloud Operations & Migrations Blog
- Capacity Blocks (capacity reservations with a defined duration)
  - Capacity Blocks for ML in the *Amazon EC2 User Guide*
  - Use Capacity Blocks for machine learning workloads

Create Auto Scaling groups from the command line using AWS CloudShell

In supported AWS Regions, you can run AWS CLI commands using AWS CloudShell for a browser-based, pre-authenticated shell that launches directly from the AWS Management Console. You can run AWS CLI commands against services using your preferred shell (Bash, PowerShell, or Z shell).

You can launch AWS CloudShell from the AWS Management Console using either one of the following two methods:

- Choose the AWS CloudShell icon on the console navigation bar. It's located to the right of the search box.
- Use the search box on the console navigation bar to search for **CloudShell** and then choose the **CloudShell** option.

When AWS CloudShell launches in a new browser window for the first time, a welcome panel displays and lists key features. After you close this panel, status updates are provided while the shell configures and forwards your console credentials. When the command prompt displays, the shell is ready for interaction.

For more information on this service, see the AWS CloudShell User Guide.

Create Auto Scaling groups with AWS CloudFormation

Amazon EC2 Auto Scaling is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your

resources and infrastructure. You create a template that describes all the AWS resources that you want (such as Auto Scaling groups), and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon EC2 Auto Scaling resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Amazon EC2 Auto Scaling and AWS CloudFormation templates

To provision and configure resources for Amazon EC2 Auto Scaling and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the *AWS CloudFormation User Guide*.

To get started creating your own stack templates for Amazon EC2 Auto Scaling, complete the following tasks:

- Create a launch template using AWS::EC2::LaunchTemplate.
- Create an Auto Scaling group using AWS::AutoScaling::AutoScalingGroup.

For a walkthrough that shows you how to deploy an Auto Scaling group behind an Application Load Balancer, see Walkthrough: Create a scaled and load-balanced application in the *AWS CloudFormation User Guide*.

You can find additional useful examples of template snippets that create Auto Scaling groups and related resources in the following sections of the *AWS CloudFormation User Guide*:

- Amazon EC2 Auto Scaling resource type reference
- Configure Amazon EC2 Auto Scaling resources with AWS CloudFormation

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation

- AWS CloudFormation User Guide
- AWS CloudFormation API Reference
- AWS CloudFormation Command Line Interface User Guide

Get instance type recommendations with AWS Compute Optimizer

AWS provides Amazon EC2 instance type recommendations to help you improve performance, save money, or both, by using features powered by AWS Compute Optimizer. You can use these recommendations to decide whether to move to a new instance type in your Auto Scaling group.

To make recommendations, Compute Optimizer analyzes your existing instance specifications and recent metric history. The compiled data is then used to recommend which Amazon EC2 instance types are best optimized to handle the existing performance workload. Recommendations are returned along with per-hour instance pricing.

> **Note**
>
> To get recommendations from Compute Optimizer, you must first opt in to Compute Optimizer. For more information, see Getting started with AWS Compute Optimizer in the *AWS Compute Optimizer User Guide*.

Contents

- Limitations
- Findings
- View recommendations
- Considerations for evaluating the recommendations

Limitations

Compute Optimizer generates recommendations for instances in Auto Scaling groups that are configured to launch and run M, C, R, T, and X instance types. However, it does not generate recommendations for -g instance types powered by AWS Graviton2 processors (e.g., C6g), and for -n instance types that have higher network bandwidth performance (e.g., M5n).

The Auto Scaling groups must also be configured to run a single instance type (i.e., no mixed instance types), must not have a scaling policy attached to them, and have the same values for desired, minimum, and maximum capacity (i.e., an Auto Scaling group with a fixed number of instances). Compute Optimizer generates recommendations for instances in Auto Scaling groups that meet *all* of these configuration requirements.

Findings

Compute Optimizer classifies its findings for Auto Scaling groups as follows:

- **Not optimized** – An Auto Scaling group is considered not optimized when Compute Optimizer has identified a recommendation that can provide better performance for your workload.
- **Optimized** – An Auto Scaling group is considered optimized when Compute Optimizer determines that the group is correctly provisioned to run your workload, based on the chosen instance type. For optimized resources, Compute Optimizer might sometimes recommend a new generation instance type.
- **None** – There are no recommendations for this Auto Scaling group. This might occur if you've been opted in to Compute Optimizer for less than 12 hours, or when the Auto Scaling group has been running for less than 30 hours, or when the Auto Scaling group or instance type is not supported by Compute Optimizer. For more information, see the Limitations section.

View recommendations

After you opt in to Compute Optimizer, you can view the findings and recommendations that it generates for your Auto Scaling groups. If you recently opted in, recommendations might not be available for up to 12 hours.

To view recommendations generated for an Auto Scaling group

1. Open the Compute Optimizer console at https://console.aws.amazon.com/compute-optimizer/.

   The Dashboard page opens.
2. Choose **View recommendations for all Auto Scaling groups**.
3. Select your Auto Scaling group.
4. Choose **View detail**.

The view changes to display up to three different instance recommendations in a preconfigured view, based on default table settings. It also provides recent CloudWatch metric data (average CPU utilization, average network in, and average network out) for the Auto Scaling group.

Determine whether you want to use one of the recommendations. Decide whether to optimize for performance improvement, for cost reduction, or for a combination of these two.

To change the instance type in your Auto Scaling group, update the launch template or update the Auto Scaling group to use a new launch configuration. Existing instances continue to use the previous configuration. To update the existing instances, terminate them so that they are replaced by your Auto Scaling group, or allow automatic scaling to gradually replace older instances with newer instances based on your termination policies.

> **Note**
>
> With the maximum instance lifetime and instance refresh features, you can also replace existing instances in your Auto Scaling group to launch new instances that use the new launch template or launch configuration. For more information, see Replace Auto Scaling instances based on maximum instance lifetime and Use an instance refresh to update instances in an Auto Scaling group.

Considerations for evaluating the recommendations

Before moving to a new instance type, consider the following:

- The recommendations don't forecast your usage. Recommendations are based on your historical usage over the most recent 14-day time period. Be sure to choose an instance type that is expected to meet your future usage needs.
- Focus on the graphed metrics to determine whether actual usage is lower than instance capacity. You can also view metric data (average, peak, percentile) in CloudWatch to further evaluate your EC2 instance recommendations. For example, notice how CPU percentage metrics change during the day and whether there are peaks that need to be accommodated. For more information, see Viewing available metrics in the *Amazon CloudWatch User Guide*.
- Compute Optimizer might supply recommendations for burstable performance instances, which are T3, T3a, and T2 instances. If you periodically burst above your baseline, make sure that you

can continue to do so based on the vCPUs of the new instance type. For more information, see CPU credits and baseline performance for burstable performance instances in the *Amazon EC2 User Guide*.

- If you've purchased a Reserved Instance, your On-Demand Instance might be billed as a Reserved Instance. Before you change your current instance type, first evaluate the impact on Reserved Instance utilization and coverage.
- Consider conversions to newer generation instances, where possible.
- When migrating to a different instance family, make sure the current instance type and the new instance type are compatible, for example, in terms of virtualization, architecture, or network type. For more information, see Compatibility for resizing instances in the *Amazon EC2 User Guide*.
- Finally, consider the performance risk rating that's provided for each recommendation. Performance risk indicates the amount of effort you might need to spend in order to validate whether the recommended instance type meets the performance requirements of your workload. We also recommend rigorous load and performance testing before and after making any changes.

Additional resources

In addition to the topics on this page, see the following resources:

- Amazon EC2 Instance Types
- AWS Compute Optimizer User Guide

Use Elastic Load Balancing; to distribute incoming application traffic in your Auto Scaling group

Elastic Load Balancing (Elastic Load Balancing) automatically distributes your incoming application traffic across all the EC2 instances that you are running. Elastic Load Balancing helps to manage incoming requests by optimally routing traffic so that no one instance is overwhelmed. To use Elastic Load Balancing with your Auto Scaling group, attach the load balancer to your Auto Scaling group. This registers the group with the load balancer, which acts as a single point of contact for all incoming web traffic to your Auto Scaling group.

When you use Elastic Load Balancing with your Auto Scaling group, it's not necessary to register individual EC2 instances with the load balancer. Instances that are launched by your Auto Scaling

group are automatically registered with the load balancer. Likewise, instances that are terminated by your Auto Scaling group are automatically deregistered from the load balancer.

After attaching a load balancer to your Auto Scaling group, you can configure your Auto Scaling group to use Elastic Load Balancing metrics (such as the Application Load Balancer request count per target) to scale the number of instances in the group as demand fluctuates.

Optionally, you can add Elastic Load Balancing health checks to your Auto Scaling group so that Amazon EC2 Auto Scaling can identify and replace unhealthy instances based on these additional health checks. Otherwise, you can create a CloudWatch alarm that notifies you if the healthy host count of the target group is lower than allowed.

Contents

- Elastic Load Balancing types
- Prepare to attach an Elastic Load Balancing load balancer
- Attach an Elastic Load Balancing load balancer to your Auto Scaling group
- Configure an Application Load Balancer or Network Load Balancer from the console
- Verify the attachment status of your load balancer
- Add an Availability Zone
- Remove an Availability Zone
- Detach a target group or Classic Load Balancer from your Auto Scaling group
- Examples for working with Elastic Load Balancing using the AWS CLI

Elastic Load Balancing types

Elastic Load Balancing provides four types of load balancers that can be used with your Auto Scaling group: Application Load Balancers, Network Load Balancers, Gateway Load Balancers, and Classic Load Balancers.

There is a key difference in how the load balancer types are configured. With Application Load Balancers, Network Load Balancers, and Gateway Load Balancers, instances are registered as targets with a target group, and you route traffic to the target group. With Classic Load Balancers, instances are registered directly with the load balancer.

Application Load Balancer

Routes and load balances at the application layer (HTTP/HTTPS), and supports path-based routing. An Application Load Balancer can route requests to ports on one or more registered targets, such as EC2 instances, in your virtual private cloud (VPC).

Network Load Balancer

Routes and load balances at the transport layer (TCP/UDP Layer-4), based on address information extracted from the Layer-4 header. Network Load Balancers can handle traffic bursts, retain the source IP of the client, and use a fixed IP for the life of the load balancer.

Gateway Load Balancer

Distributes traffic to a fleet of appliance instances. Provides scale, availability, and simplicity for third-party virtual appliances, such as firewalls, intrusion detection and prevention systems, and other appliances. Gateway Load Balancers work with virtual appliances that support the GENEVE protocol. Additional technical integration is required, so make sure to consult the user guide before choosing a Gateway Load Balancer.

Classic Load Balancer

Routes and load balances either at the transport layer (TCP/SSL), or at the application layer (HTTP/HTTPS).

To gain a deeper understanding of the different types of load balancers available, see the following resources:

- What is Elastic Load Balancing?
- What is an Application Load Balancer?
- What is a Network Load Balancer?
- What is a Gateway Load Balancer?
- What is a Classic Load Balancer?

Prepare to attach an Elastic Load Balancing load balancer

Before you attach an Elastic Load Balancing load balancer to your Auto Scaling group, you must complete the following prerequisites:

- You must have already created the load balancer and target group that is used to route traffic to your Auto Scaling group.

  There are two ways to create the load balancer and target group:

  - **Using Elastic Load Balancing** – Follow the procedures in the Elastic Load Balancing documentation to create and configure the load balancer and target group before creating the Auto Scaling group. Skip the step for registering your Amazon EC2 instances. Amazon EC2 Auto Scaling automatically takes care of registering (and deregistering) instances when you attach a target group to your Auto Scaling group. For more information, see Getting started with Elastic Load Balancing in the *Elastic Load Balancing User Guide*.
  - **Using Amazon EC2 Auto Scaling** – Create, configure, and attach the load balancer and target group with a basic configuration from the Amazon EC2 Auto Scaling console. For more information, see Configure an Application Load Balancer or Network Load Balancer from the console.
- Before creating a load balancer, know the type of load balancer that you need. For more information, see Elastic Load Balancing types.
- The load balancer and its target group must be in the same AWS account, VPC, and Region as your Auto Scaling group.
- The target group must specify a target type of `instance`. You can't specify a target type of `ip` when using an Auto Scaling group.
- If the launch template for your Auto Scaling group does not contain the correct security group to allow the necessary inbound traffic from the load balancer, you must update the launch template. The recommended rules depend on the type of load balancer and the types of backends that the load balancer uses. For example, to route traffic to web servers, allow inbound HTTP access on port 80 from the load balancer. Existing instances are not updated with the new settings when the launch template is modified. To update existing instances, you can start an instance refresh to replace the instances. For more information, see Use an instance refresh to update instances in an Auto Scaling group.
- The security groups in the launch template must also allow access from the load balancer on the correct port for Elastic Load Balancing to perform its health checks.
- When deploying virtual appliances behind a Gateway Load Balancer, the Amazon Machine Image (AMI) in the launch template must specify the ID of an AMI that supports the GENEVE protocol to allow the Auto Scaling group to exchange traffic with a Gateway Load Balancer. Also, the security groups in the launch template must allow UDP traffic on port 6081.

> **Tip**
>
> If you have bootstrapping scripts that take a while to complete, you can optionally add a launch lifecycle hook to your Auto Scaling group to delay instances from registering behind the load balancer before your bootstrap scripts have completed successfully and the applications on the instances are ready to accept traffic. You can't add a lifecycle hook when you initially create an Auto Scaling group in the Amazon EC2 Auto Scaling console. However, you can add a lifecycle hook after the group is created. For more information, see Amazon EC2 Auto Scaling lifecycle hooks.

Configure health checks for targets

You can configure health checks for your targets registered with an Elastic Load Balancing load balancer to ensure they are able to handle traffic properly. The specific steps vary based on the type of load balancer you are using. For more information, see the following resources:

- **Application Load Balancer** – See Health checks for your target groups in the *User Guide for Application Load Balancers*.
- **Network Load Balancer** – See Health checks for your target groups in the *User Guide for Network Load Balancers*.
- **Gateway Load Balancer** – See Health checks for your target groups in the *User Guide for Gateway Load Balancers*.
- **Classic Load Balancer** – See Configure health checks for your Classic Load Balancer in the *User Guide for Classic Load Balancers*.

By default, Amazon EC2 Auto Scaling does not consider an instance unhealthy and replace it if it fails the Elastic Load Balancing health checks. The default health checks for an Auto Scaling group are EC2 health checks only. For more information, see Health checks for instances in an Auto Scaling group.

To enable Amazon EC2 Auto Scaling to replace instances that are reported unhealthy by Elastic Load Balancing, you can configure your Auto Scaling group to use Elastic Load Balancing health checks. By doing so, Amazon EC2 Auto Scaling considers the instance unhealthy if it fails either the EC2 health checks or the Elastic Load Balancing health checks. If you attach multiple load balancer target groups or Classic Load Balancers to the group, all of them must report that an instance is

healthy in order for it to consider the instance healthy. If any one of them reports an instance as unhealthy, the Auto Scaling group replaces the instance, even if others report it as healthy.

For information about how to enable these health checks for your Auto Scaling group, see Attach an Elastic Load Balancing load balancer to your Auto Scaling group.

> **Note**
>
> To make sure that these health checks start as soon as possible, make sure your group's health check grace period is not set too high, but high enough for your Elastic Load Balancing health checks to determine whether a target is available to handle requests. For more information, see Set the health check grace period for an Auto Scaling group.

Attach an Elastic Load Balancing load balancer to your Auto Scaling group

This topic describes how to attach an Elastic Load Balancing load balancer to an Auto Scaling group. It also describes how to turn on Elastic Load Balancing health checks to let Amazon EC2 Auto Scaling replace instances that Elastic Load Balancing reports as unhealthy.

By default, Amazon EC2 Auto Scaling only replaces instances that are unhealthy or unreachable based on Amazon EC2 health checks. If you turn on Elastic Load Balancing health checks, Amazon EC2 Auto Scaling can replace a running instance if any of the Elastic Load Balancing load balancers you attach to the Auto Scaling group report it as unhealthy.

For a tutorial on attaching an Application Load Balancer to your Auto Scaling group, see Tutorial: Set up a scaled and load-balanced application.

> **Important**
>
> Before you continue, complete all prerequisites in the previous section.

Contents

- Attach a target group or Classic Load Balancer
- Detach a target group or Classic Load Balancer

Attach a target group or Classic Load Balancer

When you create or update an Auto Scaling group, you can attach one or more target groups or Classic Load Balancers. When you attach an Application Load Balancer, Network Load Balancer, or Gateway Load Balancer, you attach a target group rather than the load balancer itself.

Follow the steps in this section to use the console to:

- Attach a target group or Classic Load Balancer to an Auto Scaling group
- Turn on the health checks for Elastic Load Balancing

To attach an existing load balancer as you are creating a new Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your load balancer in.
3. Choose **Create Auto Scaling group**.
4. In steps 1 and 2, choose the options as desired and proceed to **Step 3: Configure advanced options**.
5. For **Load balancing**, choose **Attach to an existing load balancer**.
6. Under **Attach to an existing load balancer**, do one of the following:

   a. For Application Load Balancers, Network Load Balancers, and Gateway Load Balancers:

      Choose **Choose from your load balancer target groups**, and then choose a target group in the **Existing load balancer target groups** field.

   b. For Classic Load Balancers:

      Choose **Choose from Classic Load Balancers**, and then choose your load balancer in the **Classic Load Balancers** field.
7. (Optional) For **Health checks**, **Additional health check types**, select **Turn on Elastic Load Balancing health checks**.
8. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.

9. Proceed to create the Auto Scaling group. Your instances will be automatically registered to the load balancer after the Auto Scaling group has been created.

To attach an existing load balancer to your Auto Scaling group after it's created

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Load balancing**, **Edit**.
4. Under **Load balancing**, do one of the following:

   a. For **Application, Network or Gateway Load Balancer target groups**, select its check box and choose a target group.

   b. For **Classic Load Balancers**, select its check box and choose your load balancer.
5. Choose **Update**.

When you finish attaching the load balancer, you can optionally turn on the health checks that use it.

To turn on the Elastic Load Balancing health checks

1. On the **Details** tab, choose **Health checks**, **Edit**.
2. For **Health checks**, **Additional health check types**, select **Turn on Elastic Load Balancing health checks**.
3. For **Health check grace period**, enter the amount of time, in seconds. This is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.
4. Choose **Update**.

> **Note**
>
> You can monitor the status of the load balancer while it is being attached by using the AWS CLI. When Amazon EC2 Auto Scaling has successfully registered the instances and at least

one registered instance passes the health checks, you receive a status of `InService`. For more information, see Verify the attachment status of your load balancer.

Detach a target group or Classic Load Balancer

When you no longer need the load balancer, use the following procedure to detach it from your Auto Scaling group.

To detach a load balancer from a group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Load balancing**, **Edit**.
4. Under **Load balancing**, do one of the following:

   a. For **Application, Network or Gateway Load Balancer target groups**, choose the delete (X) icon next to the target group.

   b. For **Classic Load Balancers**, choose the delete (X) icon next to the load balancer.
5. Choose **Update**.

When you finish detaching the target group, you can turn off the Elastic Load Balancing health checks.

To turn off the Elastic Load Balancing health checks

1. On the **Details** tab, choose **Health checks**, **Edit**.
2. For **Health checks**, **Additional health check types**, deselect **Turn on Elastic Load Balancing health checks**.
3. Choose **Update**.

Configure an Application Load Balancer or Network Load Balancer from the console

Use the following procedure to create and attach an Application Load Balancer or a Network Load Balancer as you create your Auto Scaling group.

To create and attach a new load balancer as you create a new Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Choose **Create Auto Scaling group**.
3. In steps 1 and 2, choose the options as desired and proceed to **Step 3: Configure advanced options**.
4. For **Load balancing**, choose **Attach to a new load balancer**.
   a. Under **Attach to a new load balancer**, for **Load balancer type**, choose whether to create an Application Load Balancer or Network Load Balancer.
   b. For **Load balancer name**, enter a name for the load balancer, or keep the default name.
   c. For **Load balancer scheme**, choose whether to create a public internet-facing load balancer, or keep the default for an internal load balancer.
   d. For **Availability Zones and subnets**, select the public subnet for each Availability Zone in which you chose to launch your EC2 instances. (These prepopulate from step 2.).
   e. For **Listeners and routing**, update the port number for your listener (if necessary), and under **Default routing**, choose **Create a target group**. Alternatively, you can choose an existing target group from the drop-down list.
   f. If you chose **Create a target group** in the last step, for **New target group name**, enter a name for the target group, or keep the default name.
   g. To add tags to your load balancer, choose **Add tag**, and provide a tag key and value for each tag.
5. (Optional) For **Health checks**, **Additional health check types**, select **Turn on Elastic Load Balancing health checks**.
6. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.

7. Proceed to create the Auto Scaling group. Your instances will be automatically registered to the load balancer after the Auto Scaling group has been created.

> **Note**
>
> After creating your Auto Scaling group, you can use the Elastic Load Balancing console to create additional listeners. This is useful if you need to create a listener with a secure protocol, such as HTTPS, or a UDP listener. You can add more listeners to existing load balancers, as long as you use distinct ports.

Verify the attachment status of your load balancer

After you attach a load balancer, it enters the `Adding` state while registering the instances in the group. When all instances in the group are registered, it enters the `Added` state. After at least one registered instance passes the health checks, it enters the `InService` state. When the load balancer is in the `InService` state, Amazon EC2 Auto Scaling can terminate and replace any instances that are reported as unhealthy. If no registered instances pass the health checks (for example, due to a misconfigured health check), the load balancer doesn't enter the `InService` state. Amazon EC2 Auto Scaling doesn't terminate and replace the instances.

When you detach a load balancer, it enters the `Removing` state while deregistering the instances in the group. The instances remain running after they deregister. By default, connection draining (deregistration delay) is enabled for Application Load Balancers, Network Load Balancers, and Gateway Load Balancers. If connection draining is enabled, Elastic Load Balancing waits for in-flight requests to complete or for the maximum timeout to expire (whichever comes first) before it deregisters the instances.

You can verify the attachment status by using the AWS Command Line Interface (AWS CLI) or AWS SDKs. You cannot verify the attachment status from the console.

To use the AWS CLI to verify the attachment status

The following describe-traffic-sources command returns the attachment status of all traffic sources for the specified Auto Scaling group.

```
aws autoscaling describe-traffic-sources --auto-scaling-group-name my-asg
```

The example returns the ARN of the Elastic Load Balancing target group that's attached to the Auto Scaling group, along with the attachment status of the target group in the `State` element.

```
{
    "TrafficSources": [
        {
            "Identifier": "arn:aws:elasticloadbalancing:region:account-
id:targetgroup/my-targets/1234567890123456",
            "State": "InService",
            "Type": "elbv2"
        }
    ]
}
```

Add an Availability Zone

To take advantage of the safety and reliability of geographic redundancy, span your Auto Scaling group across multiple Availability Zones of the Region you are working in and attach a load balancer to distribute incoming traffic across those Availability Zones.

When one Availability Zone becomes unhealthy or unavailable, Amazon EC2 Auto Scaling launches new instances in an unaffected Availability Zone. When the unhealthy Availability Zone returns to a healthy state, Amazon EC2 Auto Scaling automatically redistributes the application instances evenly across all the Availability Zones for your Auto Scaling group. Amazon EC2 Auto Scaling does this by attempting to launch new instances in the Availability Zone with the fewest instances. If the attempt fails, however, Amazon EC2 Auto Scaling attempts to launch in other Availability Zones until it succeeds.

Elastic Load Balancing creates a load balancer node for each Availability Zone you enable for the load balancer. If you enable cross-zone load balancing for your load balancer, each load balancer node distributes traffic evenly across the registered instances in all enabled Availability Zones. If cross-zone load balancing is disabled, each load balancer node distributes requests evenly across the registered instances in its Availability Zone only.

You must specify at least one Availability Zone when you are creating your Auto Scaling group. Later, you can expand the availability of your application by adding an Availability Zone to your Auto Scaling group and enabling that Availability Zone for your load balancer (if the load balancer supports it).

Limitations

To update which Availability Zones are enabled for your load balancer, you need to be aware of the following limitations:

- When you enable an Availability Zone for your load balancer, you specify one subnet from that Availability Zone. Note that you can enable at most one subnet per Availability Zone for your load balancer.
- For internet-facing load balancers, the subnets that you specify for the load balancer must have at least eight available IP addresses.
- For Application Load Balancers, you must enable at least two Availability Zones.
- For Network Load Balancers, you cannot disable the enabled Availability Zones, but you can enable additional ones.
- For Gateway Load Balancers, you cannot disable the enabled Availability Zones, but you can enable additional ones.

Use the following procedure to expand your Auto Scaling group and load balancer to a subnet in an additional Availability Zone.

To add an Availability Zone

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Network**, **Edit**.
4. In **Subnets**, choose the subnet corresponding to the Availability Zone that you want to add to the Auto Scaling group.
5. Choose **Update**.
6. To update the Availability Zones for your load balancer so that it shares the same Availability Zones as your Auto Scaling group, complete the following steps:

   a. On the navigation pane, under **Load Balancing**, choose **Load Balancers**.

   b. Choose your load balancer.

   c. Do one of the following:

      - For Application Load Balancers and Network Load Balancers:

1. On the **Description** tab, for **Availability Zones**, choose **Edit subnets**.
2. On the **Edit subnets** page, for **Availability Zones**, select the check box for the Availability Zone to add. If there is only one subnet for that zone, it is selected. If there is more than one subnet for that zone, select one of the subnets.

- For Classic Load Balancers in a VPC:

  1. On the **Instances** tab, choose **Edit Availability Zones**.
  2. On the **Add and Remove Subnets** page, for **Available subnets**, select the subnet using its add (+) icon. The subnet is moved under **Selected subnets**.

d. Choose **Save**.

Related resources

Amazon EC2 Auto Scaling rebalances your group when you change Availability Zones. This means replacing and redistributing some instances. For more information, see Example: Distribute instances across Availability Zones.

If you have registered targets in Availability Zones that are not enabled for the load balancer, the load balancer does not route traffic to them. For more information, see How Elastic Load Balancing works in the *Elastic Load Balancing User Guide*.

Remove an Availability Zone

To remove an Availability Zone from your Auto Scaling group and load balancer, use the following procedure.

To remove an Availability Zone

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Network**, **Edit**.
4. In **Subnets**, choose the delete (X) icon for the subnet corresponding to the Availability Zone that you want to remove from the Auto Scaling group. If there is more than one subnet for that zone, choose the delete (X) icon for each one.

5. Choose **Update**.
6. To update the Availability Zones for your load balancer so that it shares the same Availability Zones as your Auto Scaling group, complete the following steps:
   a. On the navigation pane, under **Load Balancing**, choose **Load Balancers**.
   b. Choose your load balancer.
   c. Do one of the following:
      - For Application Load Balancers:
        1. On the **Description** tab, for **Availability Zones**, choose **Edit subnets**.
        2. On the **Edit subnets** page, for **Availability Zones**, clear the check box to remove the subnet for that Availability Zone.
      - For Classic Load Balancers in a VPC:
        1. On the **Instances** tab, choose **Edit Availability Zones**.
        2. On the **Add and Remove Subnets** page, for **Available subnets**, remove the subnet using its delete (-) icon. The subnet is moved under **Available subnets**.
   d. Choose **Save**.

Detach a target group or Classic Load Balancer from your Auto Scaling group

When you no longer need the load balancer, use the following procedure to detach it from your Auto Scaling group.

To detach a load balancer from a group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the **Auto Scaling groups** page.
3. On the **Details** tab, choose **Load balancing**, **Edit**.
4. Under **Load balancing**, do one of the following:

   a. For **Application, Network or Gateway Load Balancer target groups**, choose the delete (X) icon next to the target group.
   b. For **Classic Load Balancers**, choose the delete (X) icon next to the load balancer.
5. Choose **Update**.

When you finish detaching the target group, you can turn off the Elastic Load Balancing health checks.

To turn off the Elastic Load Balancing health checks

1. On the **Details** tab, choose **Health checks**, **Edit**.
2. For **Health checks**, **Additional health check types**, deselect **Turn on Elastic Load Balancing health checks**.
3. Choose **Update**.

Examples for working with Elastic Load Balancing using the AWS CLI

Use the AWS Command Line Interface (AWS CLI) to attach, detach, and describe load balancers and target groups, add and remove Elastic Load Balancing health checks, and change which Availability Zones are enabled.

This topic shows examples of AWS CLI commands that perform common tasks for Amazon EC2 Auto Scaling.

> **Important**
>
> For additional command examples, see `aws elbv2` and `aws elb` in the *AWS CLI Command Reference*.

Contents

- Attach your target group or Classic Load Balancer
- Describe your target groups or Classic Load Balancers
- Add Elastic Load Balancing health checks
- Change your Availability Zones

- Detach your target group or Classic Load Balancer
- Remove Elastic Load Balancing health checks
- Legacy commands

Attach your target group or Classic Load Balancer

Use the following create-auto-scaling-group command to create an Auto Scaling group and simultaneously attach a target group by specifying its Amazon Resource Name (ARN). The target group can be associated with an Application Load Balancer, a Network Load Balancer, or a Gateway Load Balancer.

Replace the sample values for `--auto-scaling-group-name`, `--vpc-zone-identifier`, `--min-size`, and `--max-size`. For the `--launch-template` option, replace *`my-launch-template`* and *`1`* with the name and version of a launch template for your Auto Scaling group. For the `--traffic-sources` option, replace the sample ARN with the ARN of a target group for an Application Load Balancer, Network Load Balancer, or Gateway Load Balancer.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-template LaunchTemplateName=my-launch-template,Version='1' \
  --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782" \
  --min-size 1 --max-size 5 \
  --traffic-sources "Identifier=arn:aws:elasticloadbalancing:region:account-
id:targetgroup/my-targets/12345678EXAMPLE1"
```

Use the attach-traffic-sources command to attach additional target groups to the Auto Scaling group after it's created.

The following command adds another target group to the same group.

```
aws autoscaling attach-traffic-sources --auto-scaling-group-name my-asg \
  --traffic-sources "Identifier=arn:aws:elasticloadbalancing:region:account-
id:targetgroup/my-targets/12345678EXAMPLE2"
```

Alternatively, to attach a Classic Load Balancer to your group, specify the `--traffic-sources` and `--type` options when you use **create-auto-scaling-group** or **attach-traffic-sources**, as in the following example. Replace *`my-classic-load-balancer`* with the name of a Classic Load Balancer. For the `--type` option, specify a value of **`elb`**.

```
--traffic-sources "Identifier=my-classic-load-balancer" --type elb
```

Describe your target groups or Classic Load Balancers

To describe the load balancers or target groups attached to your Auto Scaling group, use the following describe-traffic-sources command. Replace *my-asg* with the name of your group.

```
aws autoscaling describe-traffic-sources --auto-scaling-group-name my-asg
```

The example returns the ARN of the Elastic Load Balancing target groups that you attached to the Auto Scaling group.

```
{
    "TrafficSources": [
        {
            "Identifier": "arn:aws:elasticloadbalancing:region:account-
id:targetgroup/my-targets/12345678EXAMPLE1",
            "State": "InService",
            "Type": "elbv2"
        },
        {
            "Identifier": "arn:aws:elasticloadbalancing:region:account-
id:targetgroup/my-targets/12345678EXAMPLE2",
            "State": "InService",
            "Type": "elbv2"
        }
    ]
}
```

For an explanation of the `State` field in the output, see Verify the attachment status of your load balancer.

Add Elastic Load Balancing health checks

To add Elastic Load Balancing health checks to the health checks that your Auto Scaling group performs on instances, use the following update-auto-scaling-group command and specify **ELB** as the value for the `--health-check-type` option. Replace *my-asg* with the name of your group.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "ELB"
```

New instances often need time for a brief warmup before they can pass a health check. If the grace period doesn't provide enough warmup time, the instances might not appear ready to serve traffic. Amazon EC2 Auto Scaling might consider those instances unhealthy and replace them.

To update the health check grace period, use the `--health-check-grace-period` option when you use **update-auto-scaling-group**, as in the following example. Replace *300* with the number of seconds to keep new instances in service before terminating them if they're found to be unhealthy.

```
--health-check-grace-period 300
```

For more information, see Health checks for instances in an Auto Scaling group.

Change your Availability Zones

Changing your Availability Zones has some limitations that you should be aware of. For more information, see Add an Availability Zone.

To change the Availability Zones for an Application Load Balancer or Network Load Balancer

1. Before you change the Availability Zones of the load balancer, it's a good idea to first update the Availability Zones of the Auto Scaling group to verify that there is availability for your instance types in the specified zones.

   To update the Availability Zones for your Auto Scaling group, use the following update-auto-scaling-group command. Replace the sample subnet IDs with the IDs of the subnets in the Availability Zones to enable. The specified subnets replace the previously enabled subnets. Replace *my-asg* with the name of your group.

   ```
   aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
     --vpc-zone-identifier "subnet-41767929,subnet-cb663da2,subnet-8360a9e7"
   ```

2. Use the following describe-auto-scaling-groups command to verify that the instances in the new subnets have launched. If the instances have launched, you see a list of the instances and their statuses. Replace *my-asg* with the name of your group.

   ```
   aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
   ```

3. Use the following set-subnets command to specify the subnets for your load balancer. Replace the sample subnet IDs with the IDs of the subnets in the Availability Zones to enable. You can

specify only one subnet per Availability Zone. The specified subnets replace the previously enabled subnets. Replace *my-lb-arn* with the ARN of your load balancer.

```
aws elbv2 set-subnets --load-balancer-arn my-lb-arn \
  --subnets subnet-41767929 subnet-cb663da2 subnet-8360a9e7
```

To change the Availability Zones for a Classic Load Balancer

1. Before you change the Availability Zones of the load balancer, it's a good idea to first update the Availability Zones of the Auto Scaling group to verify that there is availability for your instance types in the specified zones.

   To update the Availability Zones for your Auto Scaling group, use the following update-auto-scaling-group command. Replace the sample subnet IDs with the IDs of the subnets in the Availability Zones to enable. The specified subnets replace the previously enabled subnets. Replace *my-asg* with the name of your group.

   ```
   aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
     --vpc-zone-identifier "subnet-41767929,subnet-cb663da2"
   ```

2. Use the following describe-auto-scaling-groups command to verify that the instances in the new subnets have launched. If the instances have launched, you see a list of the instances and their statuses. Replace *my-asg* with the name of your group.

   ```
   aws autoscaling describe-auto-scaling-groups --auto-scaling-group-name my-asg
   ```

3. Use the following attach-load-balancer-to-subnets command to enable a new Availability Zone for your Classic Load Balancer. Replace the sample subnet ID with the ID of the subnet for the Availability Zone to enable. Replace *my-lb* with the name of your load balancer.

   ```
   aws elb attach-load-balancer-to-subnets --load-balancer-name my-lb \
     --subnets subnet-cb663da2
   ```

   To disable an Availability Zone, use the following detach-load-balancer-from-subnets command. Replace the sample subnet ID with the ID of the subnet for the Availability Zone to disable. Replace *my-lb* with the name of your load balancer.

   ```
   aws elb detach-load-balancer-from-subnets --load-balancer-name my-lb \
   ```

```
  --subnets subnet-8360a9e7
```

Detach your target group or Classic Load Balancer

The following detach-traffic-sources command detaches a target group from your Auto Scaling group when you no longer need it.

For the `--auto-scaling-group-name` option, replace *my-asg* with the name of your group. For the `--traffic-sources` option, replace the sample ARN with the ARN of a target group for an Application Load Balancer, Network Load Balancer, or Gateway Load Balancer.

```
aws autoscaling detach-traffic-sources --auto-scaling-group-name my-asg \
  --traffic-sources "Identifier=arn:aws:elasticloadbalancing:region:account-
id:targetgroup/my-targets/1234567890123456"
```

To detach a Classic Load Balancer from your group, specify the `--traffic-sources` and `--type` options, as in the following example. Replace *my-classic-load-balancer* with the name of a Classic Load Balancer. For the `--type` option, specify a value of **elb**.

```
--traffic-sources "Identifier=my-classic-load-balancer" --type elb
```

Remove Elastic Load Balancing health checks

To remove Elastic Load Balancing health checks from your Auto Scaling group, use the following update-auto-scaling-group command and specify **EC2** as the value for the `--health-check-type` option. Replace *my-asg* with the name of your group.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "EC2"
```

For more information, see Health checks for instances in an Auto Scaling group.

Legacy commands

The following examples show how you can use legacy CLI commands to attach, detach, and describe load balancers and target groups. They remain in this document as a reference for any customers who want to use them. We continue to support the legacy CLI commands, but we recommend that you use the new "traffic sources" CLI commands, which can attach and detach

multiple traffic sources types. You can use both the legacy CLI commands and the "traffic sources" CLI commands on the same Auto Scaling group.

Attach your target group or Classic Load Balancer (legacy)

To attach your target group

The following create-auto-scaling-group command creates an Auto Scaling group with an attached target group. Specify the Amazon Resource Name (ARN) of a target group for an Application Load Balancer, Network Load Balancer, or Gateway Load Balancer.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-template LaunchTemplateName=my-launch-template,Version='1' \
  --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782" \
  --target-group-arns "arn:aws:elasticloadbalancing:region:account-id:targetgroup/my-
targets/1234567890123456" \
  --min-size 1 --max-size 5
```

The following attach-load-balancer-target-groups command attaches a target group to an existing Auto Scaling group.

```
aws autoscaling attach-load-balancer-target-groups --auto-scaling-group-name my-asg \
  --target-group-arns "arn:aws:elasticloadbalancing:region:account-id:targetgroup/my-
targets/1234567890123456"
```

To attach your Classic Load Balancer

The following create-auto-scaling-group command creates an Auto Scaling group with an attached Classic Load Balancer.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
  --launch-configuration-name my-launch-config \
  --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782" \
  --load-balancer-names "my-load-balancer" \
  --min-size 1 --max-size 5
```

The following attach-load-balancers command attaches the specified Classic Load Balancer to an existing Auto Scaling group.

```
aws autoscaling attach-load-balancers --auto-scaling-group-name my-asg \
  --load-balancer-names my-lb
```

Describe your target group or Classic Load Balancer (legacy)

To describe target groups

To describe the target groups associated with an Auto Scaling group, use the describe-load-balancer-target-groups command. The following example lists the target groups for *my-asg*.

```
aws autoscaling describe-load-balancer-target-groups --auto-scaling-group-name my-asg
```

To describe Classic Load Balancers

To describe the Classic Load Balancers associated with an Auto Scaling group, use the describe-load-balancers command. The following example lists the Classic Load Balancers for *my-asg*.

```
aws autoscaling describe-load-balancers --auto-scaling-group-name my-asg
```

Detach your target group or Classic Load Balancer (legacy)

To detach a target group

The following detach-load-balancer-target-groups command detaches a target group from your Auto Scaling group when you no longer need it.

```
aws autoscaling detach-load-balancer-target-groups --auto-scaling-group-name my-asg \
  --target-group-arns "arn:aws:elasticloadbalancing:region:account-id:targetgroup/my-
targets/1234567890123456"
```

To detach a Classic Load Balancer

The following detach-load-balancers command detaches a Classic Load Balancer from your Auto Scaling group when you no longer need it.

```
aws autoscaling detach-load-balancers --auto-scaling-group-name my-asg \
  --load-balancer-names my-lb
```

Manage traffic flow with a VPC Lattice target group

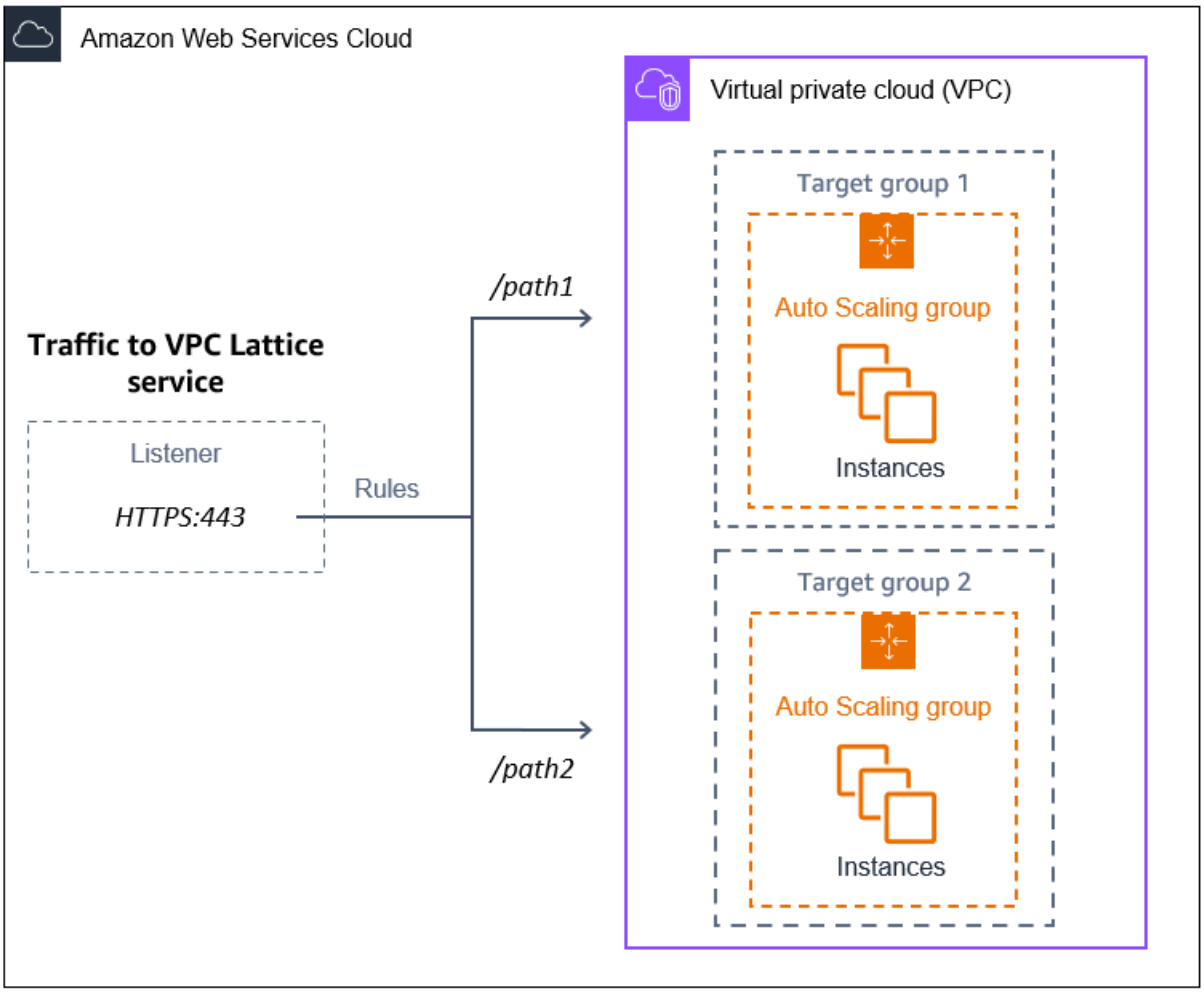
You can use Amazon VPC Lattice to manage the flow of traffic and API calls between your applications and services that run on separate resources, such as Auto Scaling groups or Lambda

functions. VPC Lattice is an application networking service that lets you connect, secure, and monitor all your services across multiple accounts and virtual private clouds (VPCs). To learn more about VPC Lattice, see What is VPC Lattice?

To get started with VPC Lattice, first create the necessary VPC Lattice resources that enable resources in a VPC associated with a service network to connect to each other. These resources include the services, listeners, listener rules, and target groups.

To associate an Auto Scaling group to a VPC Lattice service, create a target group for the service that routes requests to instances registered by instance ID, and add a listener to the service that sends requests to the target group. Then, attach the target group to your Auto Scaling group. Amazon EC2 Auto Scaling automatically registers the EC2 instances as targets with the target group. Later, when Amazon EC2 Auto Scaling needs to terminate an instance, it automatically deregisters the instance from the target group before termination.

After you attach the target group, it's the entry point for all incoming requests to your Auto Scaling group. As the example in the following diagram shows, incoming requests can then be routed to the appropriate target group using listener rules specified for a VPC Lattice service.



When traffic is routed through VPC Lattice to your Auto Scaling group, VPC Lattice balances requests among the instances in the group using round robin load balancing. VPC Lattice also can monitor the health of its registered instances and route traffic only to healthy instances.

To keep your instances available for incoming requests, you can optionally add VPC Lattice health checks to your Auto Scaling group. This way, if one of the EC2 instances fails, your Auto Scaling group automatically launches a new instance to replace it. The behavior of the VPC Lattice health checks is similar to the behavior of the Elastic Load Balancing health checks. The default health checks for an Auto Scaling group are EC2 health checks only.

To learn more about VPC Lattice, see Simplify Service-to-Service Connectivity, Security, and Monitoring with Amazon VPC Lattice – Now Generally Available on the AWS Blog.

Contents

- Prepare to attach a VPC Lattice target group to your Auto Scaling group

- Attach a VPC Lattice target group to your Auto Scaling group
- Verify the attachment status of your VPC Lattice target group

Prepare to attach a VPC Lattice target group to your Auto Scaling group

Before you attach a VPC Lattice target group to your Auto Scaling group, you must complete the following prerequisites:

- You must have already created a VPC Lattice service network, service, listener, and target group. For more information, see the following topics in the *VPC Lattice User Guide*:
  - Service networks
  - Services
  - Listeners
  - Target groups
- The target group must be in the same AWS account, VPC, and Region as your Auto Scaling group.
- The target group must specify a target type of `instance`. You can't specify a target type of `ip` when using an Auto Scaling group.
- You must have sufficient IAM permissions to attach the target group to the Auto Scaling group. The following example policy shows the minimum required permissions that are necessary to attach and detach target groups.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:AttachTrafficSources",
                "autoscaling:DetachTrafficSources",
                "autoscaling:DescribeTrafficSources",
                "vpc-lattice:RegisterTargets",
                "vpc-lattice:DeregisterTargets"
            ],
            "Resource": "*"
        }
    ]
}
```

- If the launch template for your Auto Scaling group does not contain the correct settings for VPC Lattice, such as a compatible security group, you must update the launch template. Existing instances are not updated with the new settings when the launch template is modified. To update existing instances, you can start an instance refresh to replace the instances. For more information, see Use an instance refresh to update instances in an Auto Scaling group.
- Before enabling the VPC Lattice health checks on your Auto Scaling group, you can configure an application-based health check to verify that your application is responding as expected. For more information, see Health checks for your target groups in the *VPC Lattice User Guide*.

Security groups: Inbound and outbound rules

Security groups act as a firewall for associated EC2 instances, controlling both inbound and outbound traffic at the instance level.

> **Note**
>
> Network configuration is sufficiently complex that we strongly recommend that you create a new security group for use with VPC Lattice. It also makes it easier for AWS Support to help you if you need to contact them. The following sections are based on the assumption that you follow this recommendation.
> To learn more about creating security groups for VPC Lattice that you can use with your Auto Scaling group, see Control traffic using security groups in the *VPC Lattice User Guide*.
> To troubleshoot issues with traffic flow, consult the *VPC Lattice User Guide* for further information.

For information about how to create a security group, see Create a security group in the *Amazon EC2 User Guide* and use the following table to determine what options to select.

Option	Value
Name	A name that's easy for you to remember.
Description	A description to help you identify the security group.

Option	Value	
VPC	The same VPC as the Auto Scaling group.	

Inbound rules

When you create a security group, it has no inbound rules. No inbound traffic originating from clients within a VPC Lattice service network to your instance is allowed until you add inbound rules to the security group.

To allow clients within a VPC Lattice service network to connect to instances in your Auto Scaling group, the security group for your Auto Scaling group must be set up correctly. In this case, give it an inbound rule to allow traffic from the name of the AWS managed prefix list for VPC Lattice, instead of a specific IP address. The VPC Lattice prefix list is a range of IP addresses used by VPC Lattice in CIDR notation. For more information, see Work with AWS-managed prefix lists in the *Amazon VPC User Guide*.

For information about how to add rules to a security group, see Add rules to your security group in the *Amazon VPC User Guide* and use the following table to determine what options to select.

Option	Value	
HTTP rule	Type: HTTP Source: com.amazonaws.*region*.vpc-lattice	
HTTPS rule	Type: HTTPS Source: com.amazonaws.*region*.vpc-lattice	

The security group is stateful: it allows traffic from clients within the VPC Lattice service network to instances in your Auto Scaling group, and then sends the response back to the client it previously left.

Outbound rules

By default, a security group includes an outbound rule that allows all outbound traffic. You can optionally remove this default rule and add an outbound rule to accommodate specific security needs.

Limitations

- Mixed instances groups are not supported. If you try to attach a VPC Lattice target group to an Auto Scaling group that has a mixed instances policy, you receive the error message **Currently, Auto Scaling Groups with mixed instances cannot be integrated with a VPC Lattice service**. This is because the load balancing algorithm evenly distributes load onto all available resources and assumes that instances are similar enough to handle equal loads.

Attach a VPC Lattice target group to your Auto Scaling group

This topic describes how to attach a VPC Lattice target group to an Auto Scaling group. It also describes how to turn on VPC Lattice health checks to let Amazon EC2 Auto Scaling replace instances that VPC Lattice reports as unhealthy.

By default, Amazon EC2 Auto Scaling only replaces instances that are unhealthy or unreachable based on Amazon EC2 health checks. If you turn on VPC Lattice health checks, Amazon EC2 Auto Scaling can replace a running instance if any of the VPC Lattice target groups you attach to the Auto Scaling group report it as unhealthy. For more information, see Health checks for instances in an Auto Scaling group.

> **Important**
>
> Before you continue, complete all prerequisites in the previous section.

Attach a VPC Lattice target group

You can attach one or more target groups to an Auto Scaling group when you create or update the group.

Console

Follow the steps in this section to use the console to:

- Attach a VPC Lattice target group to an Auto Scaling group
- Turn on the health checks for VPC Lattice

To attach a VPC Lattice target group to a new Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the AWS Region that you created your target group in.
3. Choose **Create Auto Scaling group**.
4. In steps 1 and 2, choose your desired options and proceed to **Step 3: Configure advanced options**.
5. For **VPC Lattice integration options**, choose **Attach to VPC Lattice service**.
6. Under **Choose VPC Lattice target group**, choose your target group.
7. (Optional) For **Health checks**, **Additional health check types**, select **Turn on VPC Lattice health checks**.
8. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.
9. Proceed to create the Auto Scaling group. Your instances will be automatically registered to the VPC Lattice target group after the Auto Scaling group has been created.

To attach a VPC Lattice target group to an existing Auto Scaling group

Use the following procedure to attach a target group for a service to an existing Auto Scaling group.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.
2. Select the check box next to your Auto Scaling group.

   A split pane opens up in the bottom of the page.
3. On the **Details** tab, choose **VPC Lattice integration options**, **Edit**.
4. Under **VPC Lattice integration options**, choose **Attach to VPC Lattice service**.

5. Under **Choose VPC Lattice target group**, choose your target group.
6. Choose **Update**.

When you finish attaching the target group, you can optionally turn on the health checks that use it.

To turn on the VPC Lattice health checks

1. On the **Details** tab, choose **Health checks**, **Edit**.
2. For **Health checks**, **Additional health check types**, select **Turn on VPC Lattice health checks**.
3. For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the `InService` state. For more information, see Set the health check grace period for an Auto Scaling group.
4. Choose **Update**.

AWS CLI

Follow the steps in this section to use the AWS CLI to:

- Attach a VPC Lattice target group to an Auto Scaling group
- Turn on the health checks for VPC Lattice

To attach a VPC Lattice target group to an Auto Scaling group

Use the following create-auto-scaling-group command to create an Auto Scaling group and simultaneously attach a VPC Lattice target group by specifying its Amazon Resource Name (ARN).

Replace the sample values for `--auto-scaling-group-name`, `--vpc-zone-identifier`, `--min-size`, and `--max-size`. For the `--launch-template` option, replace *`my-launch-template`* and *`1`* with the name and version of the launch template that you created for instances registered to a VPC Lattice target group. For the `--traffic-sources` option, replace the sample ARN with the ARN of your VPC Lattice target group.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg \
```

```
  --launch-template LaunchTemplateName=my-launch-template,Version='1' \
  --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782" \
  --min-size 1 --max-size 5 \
  --traffic-sources "Identifier=arn:aws:vpc-lattice:region:account-id:targetgroup/
tg-0e2f2665eEXAMPLE"
```

Use the following attach-traffic-sources command to attach a VPC Lattice target group to an Auto Scaling group after it's already created.

```
aws autoscaling attach-traffic-sources --auto-scaling-group-name my-asg \
  --traffic-sources "Identifier=arn:aws:vpc-lattice:region:account-id:targetgroup/
tg-0e2f2665eEXAMPLE"
```

To turn on the health checks for VPC Lattice

If you have configured an application-based health check for your **VPC Lattice** target group, you can turn on these health checks. Use the create-auto-scaling-group or update-auto-scaling-group command with the `--health-check-type` option and a value of **`VPC_LATTICE`**. To specify the grace period for the health checks performed by your Auto Scaling group, include the `--health-check-grace-period` option and provide its value in seconds.

```
--health-check-type "VPC_LATTICE" --health-check-grace-period 60
```

Detach a VPC Lattice target group

If you no longer need to use VPC Lattice, use the following procedure to detach the target group from your Auto Scaling group.

Console

Follow the steps in this section to use the console to:

- Detach a VPC Lattice target group from an Auto Scaling group
- Turn off the health checks for VPC Lattice

To detach a VPC Lattice target group from an Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/, and choose **Auto Scaling Groups** from the navigation pane.

2. Select the check box next to an existing group.

   A split pane opens up in the bottom of the page.
3. On the **Details** tab, choose **VPC Lattice integration options**, **Edit**.
4. Under **VPC Lattice integration options**, choose the delete (X) icon next to the target group.
5. Choose **Update**.

When you finish detaching the target group, you can turn off the VPC Lattice health checks.

To turn off the VPC Lattice health checks

1. On the **Details** tab, choose **Health checks**, **Edit**.
2. For **Health checks**, **Additional health check types**, deselect **Turn on VPC Lattice health checks**.
3. Choose **Update**.

AWS CLI

Follow the steps in this section to use the AWS CLI to:

- Detach a VPC Lattice target group from an Auto Scaling group
- Turn off the health checks for VPC Lattice

Use the detach-traffic-sources command to detach a target group from your Auto Scaling group when you no longer need it.

```
aws autoscaling detach-traffic-sources --auto-scaling-group-name my-asg \
  --traffic-sources "Identifier=arn:aws:vpc-lattice:region:account-id:targetgroup/
tg-0e2f2665eEXAMPLE"
```

To update the health checks on an Auto Scaling group so that it no longer uses VPC Lattice health checks, use the update-auto-scaling-group command. Include the `--health-check-type` option and a value of **EC2**.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-asg \
  --health-check-type "EC2"
```

Verify the attachment status of your VPC Lattice target group

After you attach a VPC Lattice target group to an Auto Scaling group, it enters the `Adding` state while registering the instances in the group. When all instances in the group are registered, it enters the `Added` state. After at least one registered instance passes the health checks, it enters the `InService` state. When the target group is in the `InService` state, Amazon EC2 Auto Scaling can terminate and replace any instances that are reported as unhealthy. If no registered instances pass the health checks (for example, due to a misconfigured health check), the target group doesn't enter the `InService` state. Amazon EC2 Auto Scaling doesn't terminate and replace the instances.

When you detach a target group for a service, it enters the `Removing` state while deregistering the instances in the group. The instances remain running after they deregister. By default, connection draining (deregistration delay) is enabled. If connection draining is enabled, VPC Lattice waits for in-flight requests to complete or for the maximum timeout to expire (whichever comes first) before it deregisters the instances.

You can verify the attachment status by using the AWS Command Line Interface (AWS CLI) or AWS SDKs. You cannot verify the attachment status from the console.

To use the AWS CLI to verify the attachment status

The following describe-traffic-sources command returns the attachment status of all traffic sources for the specified Auto Scaling group.

```
aws autoscaling describe-traffic-sources --auto-scaling-group-name my-asg
```

The example returns the ARN of the VPC Lattice target group that's attached to the Auto Scaling group, along with the attachment status of the target group in the `State` element.

```
{
    "TrafficSources": [
        {
            "Identifier": "arn:aws:vpc-lattice:region:account-
id:targetgroup/tg-0e2f2665eEXAMPLE",
            "State": "InService",
            "Type": "vpc-lattice"
        }
    ]
}
```

Use EventBridge to handle Auto Scaling events

Amazon EventBridge, formerly called CloudWatch Events, helps you set up event-driven rules that monitor resources and initiate target actions that use other AWS services.

Events from Amazon EC2 Auto Scaling are delivered to EventBridge in near real time. You can establish EventBridge rules that invoke programmatic actions and notifications in response to a variety of these events. For example, while instances are in the process of launching or terminating, you can invoke an AWS Lambda function to perform a preconfigured task.

Targets of EventBridge rules can include AWS Lambda functions, Amazon SNS topics, API destinations, event buses in other AWS accounts, and many more. For information about supported targets, see Amazon EventBridge targets in the *Amazon EventBridge User Guide*.

Get started by creating EventBridge rules with an example using an Amazon SNS topic and an EventBridge rule. Then, when a user starts an instance refresh, Amazon SNS notifies you by email whenever a checkpoint is reached. For more information, see Create EventBridge rules for instance refresh events.

Contents

- Amazon EC2 Auto Scaling event reference
- Warm pool example events and patterns
- Use Amazon EventBridge rules to automate actions

Amazon EC2 Auto Scaling event reference

Using Amazon EventBridge, you can create *rules* that match incoming *events* and route them to *targets* for processing.

Contents

- Lifecycle action events
- Successful scaling events
- Unsuccessful scaling events
- Instance refresh events

Lifecycle action events

When you add lifecycle hooks to your Auto Scaling group, Amazon EC2 Auto Scaling sends events to EventBridge when an instance transitions into a wait state. Events are produced on a best-effort basis.

Event types

- Scale-out lifecycle action
- Scale-in lifecycle action

Scale-out lifecycle action

The following example event shows that Amazon EC2 Auto Scaling moved an instance to a `Pending:Wait` state due to a launch lifecycle hook.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance-launch Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "LifecycleActionToken": "87654321-4321-4321-4321-210987654321",
    "AutoScalingGroupName": "my-asg",
    "LifecycleHookName": "my-lifecycle-hook",
    "EC2InstanceId": "i-1234567890abcdef0",
    "LifecycleTransition": "autoscaling:EC2_INSTANCE_LAUNCHING",
    "NotificationMetadata": "additional-info",
    "Origin": "EC2",
    "Destination": "AutoScalingGroup"
  }
}
```

Scale-in lifecycle action

The following example event shows that Amazon EC2 Auto Scaling moved an instance to a `Terminating:Wait` state due to a termination lifecycle hook.

> **Important**
>
> When an Auto Scaling group returns instances to a warm pool on scale in, returning instances to the warm pool can also generate `EC2 Instance-terminate Lifecycle Action` events. Events that are delivered when an instance moves to the wait state on scale in have `WarmPool` as the value for `Destination`. For more information, see Instance reuse policy.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance-terminate Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "LifecycleActionToken":"87654321-4321-4321-4321-210987654321",
    "AutoScalingGroupName":"my-asg",
    "LifecycleHookName":"my-lifecycle-hook",
    "EC2InstanceId":"i-1234567890abcdef0",
    "LifecycleTransition":"autoscaling:EC2_INSTANCE_TERMINATING",
    "NotificationMetadata":"additional-info",
    "Origin": "AutoScalingGroup",
    "Destination": "EC2"
  }
}
```

Successful scaling events

The following examples show the event types for successful scaling events. Events are produced on a best-effort basis.

Event types

- Successful scale-out event
- Successful scale-in event

Successful scale-out event

The following example event shows that Amazon EC2 Auto Scaling successfully launched an instance.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance Launch Successful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn",
    "instance-arn"
  ],
  "detail": {
    "StatusCode": "InProgress",
    "Description": "Launching a new EC2 instance: i-12345678",
    "AutoScalingGroupName": "my-asg",
    "ActivityId": "87654321-4321-4321-4321-210987654321",
    "Details": {
      "Availability Zone": "us-west-2b",
      "Subnet ID": "subnet-12345678"
    },
    "RequestId": "12345678-1234-1234-1234-123456789012",
    "StatusMessage": "",
    "EndTime": "yyyy-mm-ddThh:mm:ssZ",
    "EC2InstanceId": "i-1234567890abcdef0",
    "StartTime": "yyyy-mm-ddThh:mm:ssZ",
    "Cause": "description-text",
    "Origin": "EC2",
    "Destination": "AutoScalingGroup"
  }
}
```

Successful scale-in event

The following example event shows that Amazon EC2 Auto Scaling successfully terminated an instance.

> **Important**
>
> When an Auto Scaling group returns instances to a warm pool on scale in, returning instances to the warm pool can also generate `EC2 Instance Terminate Successful` events. Events that are delivered when an instance successfully returns to the warm pool have `WarmPool` as the value for `Destination`. For more information, see Instance reuse policy.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance Terminate Successful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn",
    "instance-arn"
  ],
  "detail": {
    "StatusCode": "InProgress",
    "Description": "Terminating EC2 instance: i-12345678",
    "AutoScalingGroupName": "my-asg",
    "ActivityId": "87654321-4321-4321-4321-210987654321",
    "Details": {
      "Availability Zone": "us-west-2b",
      "Subnet ID": "subnet-12345678"
    },
    "RequestId": "12345678-1234-1234-1234-123456789012",
    "StatusMessage": "",
    "EndTime": "yyyy-mm-ddThh:mm:ssZ",
    "EC2InstanceId": "i-1234567890abcdef0",
    "StartTime": "yyyy-mm-ddThh:mm:ssZ",
    "Cause": "description-text",
    "Origin": "AutoScalingGroup",
```

```
    "Destination": "EC2"
  }
}
```

Unsuccessful scaling events

The following examples show the event types for unsuccessful scaling events. Events are produced on a best-effort basis.

Event types

- Unsuccessful scale-out event
- Unsuccessful scale-in event

Unsuccessful scale-out event

The following example event shows that Amazon EC2 Auto Scaling failed to launch an instance.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance Launch Unsuccessful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn",
    "instance-arn"
  ],
  "detail": {
    "StatusCode": "Failed",
    "AutoScalingGroupName": "my-asg",
    "ActivityId": "87654321-4321-4321-4321-210987654321",
    "Details": {
      "Availability Zone": "us-west-2b",
      "Subnet ID": "subnet-12345678"
    },
    "RequestId": "12345678-1234-1234-1234-123456789012",
    "StatusMessage": "message-text",
    "EndTime": "yyyy-mm-ddThh:mm:ssZ",
    "EC2InstanceId": "i-1234567890abcdef0",
```

```
    "StartTime": "yyyy-mm-ddThh:mm:ssZ",
    "Cause": "description-text",
    "Origin": "EC2",
    "Destination": "AutoScalingGroup"
  }
}
```

Unsuccessful scale-in event

The following example event shows that Amazon EC2 Auto Scaling failed to terminate an instance.

> **Important**
>
> When an Auto Scaling group returns instances to a warm pool on scale in, failing to return instances to the warm pool can also generate `EC2 Instance Terminate Unsuccessful` events. Events that are delivered when an instance fails to return to the warm pool have `WarmPool` as the value for `Destination`. For more information, see Instance reuse policy.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance Terminate Unsuccessful",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn",
    "instance-arn"
  ],
  "detail": {
    "StatusCode": "Failed",
    "AutoScalingGroupName": "my-asg",
    "ActivityId": "87654321-4321-4321-4321-210987654321",
    "Details": {
      "Availability Zone": "us-west-2b",
      "Subnet ID": "subnet-12345678"
    },
    "RequestId": "12345678-1234-1234-1234-123456789012",
    "StatusMessage": "message-text",
```

```
    "EndTime": "yyyy-mm-ddThh:mm:ssZ",
    "EC2InstanceId": "i-1234567890abcdef0",
    "StartTime": "yyyy-mm-ddThh:mm:ssZ",
    "Cause": "description-text",
    "Origin": "AutoScalingGroup",
    "Destination": "EC2"
  }
}
```

Instance refresh events

The following examples show events for the instance refresh feature. Events are produced on a best-effort basis.

Event types

- Checkpoint reached
- Instance refresh started
- Instance refresh succeeded
- Instance refresh failed
- Instance refresh cancelled
- Instance refresh rollback started
- Instance refresh rollback succeeded
- Instance refresh rollback failed

Checkpoint reached

When the number of instances that have been replaced reaches the percentage threshold defined for the checkpoint, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Checkpoint Reached",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
```

```
  ],
  "detail": {
    "InstanceRefreshId": "ab00cf8f-9126-4f3c-8010-dbb8cad6fb86",
    "AutoScalingGroupName": "my-asg",
    "CheckpointPercentage": "50",
    "CheckpointDelay": "300"
  }
}
```

Instance refresh started

When the status of an instance refresh changes to `InProgress`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Started",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Instance refresh succeeded

When the status of an instance refresh changes to `Successful`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Succeeded",
  "source": "aws.autoscaling",
  "account": "123456789012",
```

```
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Instance refresh failed

When the status of an instance refresh changes to `Failed`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Failed",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Instance refresh cancelled

When the status of an instance refresh changes to `Cancelled`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Cancelled",
```

```
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Instance refresh rollback started

When the status of an instance refresh changes to `RollbackInProgress`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Rollback Started",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Instance refresh rollback succeeded

When the status of an instance refresh changes to `RollbackSuccessful`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
```

```
  "detail-type": "EC2 Auto Scaling Instance Refresh Rollback Succeeded",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Instance refresh rollback failed

When the status of an instance refresh changes to `Failed`, Amazon EC2 Auto Scaling sends the following event.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Auto Scaling Instance Refresh Rollback Failed",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "yyyy-mm-ddThh:mm:ssZ",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "InstanceRefreshId": "c613620e-07e2-4ed2-a9e2-ef8258911ade",
    "AutoScalingGroupName": "my-asg"
  }
}
```

Warm pool example events and patterns

Amazon EC2 Auto Scaling supports several predefined patterns in Amazon EventBridge. This simplifies how an event pattern is created. You select field values on a form, and EventBridge generates the pattern for you. At this time, Amazon EC2 Auto Scaling doesn't support predefined patterns for any events that are emitted by an Auto Scaling group with a warm pool. You must

enter the pattern as a JSON object. This section and the Create EventBridge rules for warm pool events topic show you how to use an event pattern to select events and send them to targets.

To create EventBridge rules that filter for warm pool-related events that Amazon EC2 Auto Scaling sends to EventBridge, include the `Origin` and `Destination` fields from the `detail` section of the event.

The values of `Origin` and `Destination` can be the following:

`EC2` | `AutoScalingGroup` | `WarmPool`

Contents

- Example events
- Example event patterns

Example events

When you add lifecycle hooks to your Auto Scaling group, Amazon EC2 Auto Scaling sends events to EventBridge when an instance transitions into a wait state. For more information, see Use lifecycle hooks with a warm pool in Auto Scaling group.

This section includes examples of these events when your Auto Scaling group has a warm pool. Events are emitted on a best-effort basis.

> **Note**
>
> For events that Amazon EC2 Auto Scaling sends to EventBridge when scaling is successful, see Successful scaling events. For events when scaling is unsuccessful, see Unsuccessful scaling events.

Event examples

- Scale-out lifecycle action
- Scale-in lifecycle action

Scale-out lifecycle action

Events that are delivered when an instance transitions into a wait state for scale-out events have `EC2 Instance-launch Lifecycle Action` as the value for `detail-type`. In the `detail`

object, the values for the `Origin` and `Destination` attributes show where the instance is coming from and where it's going.

In this example scale-out event, a new instance launches and its state changes to `Warmed:Pending:Wait` because it's added to the warm pool. For more information, see Lifecycle state transitions for instances in a warm pool.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance-launch Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2021-01-13T00:12:37.214Z",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "LifecycleActionToken": "71514b9d-6a40-4b26-8523-05e7eEXAMPLE",
    "AutoScalingGroupName": "my-asg",
    "LifecycleHookName": "my-launch-lifecycle-hook",
    "EC2InstanceId": "i-1234567890abcdef0",
    "LifecycleTransition": "autoscaling:EC2_INSTANCE_LAUNCHING",
    "NotificationMetadata": "additional-info",
    "Origin": "EC2",
    "Destination": "WarmPool"
  }
}
```

In this example scale-out event, the state of the instance changes to `Pending:Wait` because it's added to the Auto Scaling group from the warm pool. For more information, see Lifecycle state transitions for instances in a warm pool.

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance-launch Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2021-01-19T00:35:52.359Z",
  "region": "us-west-2",
```

```
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "LifecycleActionToken": "19cc4d4a-e450-4d1c-b448-0de67EXAMPLE",
    "AutoScalingGroupName": "my-asg",
    "LifecycleHookName": "my-launch-lifecycle-hook",
    "EC2InstanceId": "i-1234567890abcdef0",
    "LifecycleTransition": "autoscaling:EC2_INSTANCE_LAUNCHING",
    "NotificationMetadata": "additional-info",
    "Origin": "WarmPool",
    "Destination": "AutoScalingGroup"
  }
}
```

Scale-in lifecycle action

Events that are delivered when an instance transitions into a wait state for scale-in events have `EC2 Instance-terminate Lifecycle Action` as the value for `detail-type`. In the `detail` object, the values for the `Origin` and `Destination` attributes show where the instance is coming from and where it's going.

In this example scale-in event, the state of an instance changes to `Warmed:Pending:Wait` because it's returned to the warm pool. For more information, see [Lifecycle state transitions for instances in a warm pool](#).

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789012",
  "detail-type": "EC2 Instance-terminate Lifecycle Action",
  "source": "aws.autoscaling",
  "account": "123456789012",
  "time": "2022-03-28T00:12:37.214Z",
  "region": "us-west-2",
  "resources": [
    "auto-scaling-group-arn"
  ],
  "detail": {
    "LifecycleActionToken": "42694b3d-4b70-6a62-8523-09a1eEXAMPLE",
    "AutoScalingGroupName": "my-asg",
    "LifecycleHookName": "my-termination-lifecycle-hook",
    "EC2InstanceId": "i-1234567890abcdef0",
    "LifecycleTransition": "autoscaling:EC2_INSTANCE_TERMINATING",
```

```
    "NotificationMetadata": "additional-info",
    "Origin": "AutoScalingGroup",
    "Destination": "WarmPool"
  }
}
```

Example event patterns

The preceding section provides example events emitted by Amazon EC2 Auto Scaling.

EventBridge event patterns have the same structure as the events that they match. The pattern quotes the fields that you want to match and provides the values that you're looking for.

The following fields in the event form the event pattern that is defined in the rule to invoke an action:

`"source": "aws.autoscaling"`

Identifies that the event is from Amazon EC2 Auto Scaling.

`"detail-type": "EC2 Instance-launch Lifecycle Action"`

Identifies the event type.

`"Origin": "EC2"`

Identifies where the instance is coming from.

`"Destination": "WarmPool"`

Identifies where the instance is going to.

Use the following sample event pattern to capture all `EC2 Instance-launch Lifecycle Action` events that are associated with instances entering the warm pool.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-launch Lifecycle Action" ],
  "detail": {
      "Origin": [ "EC2" ],
      "Destination": [ "WarmPool" ]
   }
```

```
}
```

Use the following sample event pattern to capture all `EC2 Instance-launch Lifecycle Action` events that are associated with instances leaving the warm pool because of a scale-out event.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-launch Lifecycle Action" ],
  "detail": {
      "Origin": [ "WarmPool" ],
      "Destination": [ "AutoScalingGroup" ]
   }
}
```

Use the following sample event pattern to capture all `EC2 Instance-launch Lifecycle Action` events that are associated with instances launching directly into the Auto Scaling group.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-launch Lifecycle Action" ],
  "detail": {
      "Origin": [ "EC2" ],
      "Destination": [ "AutoScalingGroup" ]
   }
}
```

Use the following sample event pattern to capture all `EC2 Instance-terminate Lifecycle Action` events that are associated with instances returning to the warm pool on scale in.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-terminate Lifecycle Action" ],
  "detail": {
      "Origin": [ "AutoScalingGroup" ],
      "Destination": [ "WarmPool" ]
   }
}
```

Use the following sample event pattern to capture all events that are associated with `EC2 Instance-launch Lifecycle Action`, regardless of the origin or destination.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-launch Lifecycle Action" ]
}
```

Use Amazon EventBridge rules to automate actions

When an event is emitted by Amazon EC2 Auto Scaling, an event notification is sent to Amazon EventBridge as a JSON file. You can write an EventBridge rule to automate what actions to take when an event pattern matches the rule. If EventBridge detects an event pattern that matches a pattern defined in a rule, EventBridge invokes the target (or targets) specified in the rule.

You can use the example procedures in this section as a starting point.

You may also find the following documentation useful.

- To perform custom actions on instances as they are launching or before they are terminated using a Lambda function, see Tutorial: Configure a lifecycle hook that invokes a Lambda function.
- To invoke a Lambda function on API calls logged with CloudTrail, see Tutorial: Log AWS API calls using EventBridge in the *Amazon EventBridge User Guide*.
- For more information about how to create event rules, see Creating Amazon EventBridge rules that react to events in the *Amazon EventBridge User Guide*.

Topics

- Create EventBridge rules for instance refresh events
- Create EventBridge rules for warm pool events

Create EventBridge rules for instance refresh events

The following example creates an EventBridge rule to send an email notification. It does this each time that your Auto Scaling group emits an event when a checkpoint is reached during an instance refresh. The procedure for setting up email notifications using Amazon SNS is included. To use Amazon SNS to send email notifications, you must first create a *topic* and then subscribe your email addresses to the topic.

For more information about the instance refresh feature, see Use an instance refresh to update instances in an Auto Scaling group.

Create an Amazon SNS topic

An SNS topic is a logical access point, a communication channel that your Auto Scaling group uses to send the notifications. You create a topic by specifying a name for your topic.

Topic names must meet the following requirements:

- Have 1-256 characters
- Contain uppercase and lowercase ASCII letters, numbers, underscores, or hyphens

For more information, see Creating an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

Subscribe to the Amazon SNS topic

To receive the notifications that your Auto Scaling group sends to the topic, you must subscribe an endpoint to the topic. In this procedure, for **Endpoint**, specify the email address where you want to receive the notifications from Amazon EC2 Auto Scaling.

For more information, see Subscribing to an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

Confirm your Amazon SNS subscription

Amazon SNS sends a confirmation email to the email address you specified in the previous step.

Make sure that you open the email from AWS Notifications and choose the link to confirm the subscription before you continue with the next step.

You will receive an acknowledgment message from AWS. Amazon SNS is now configured to receive notifications and send the notification as an email to the email address that you specified.

Route events to your Amazon SNS topic

Create a rule that matches selected events and routes them to your Amazon SNS topic to notify subscribed email addresses.

To create a rule that sends notifications to your Amazon SNS topic

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.

2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. For **Define rule detail**, do the following:
   a. Enter a **Name** for the rule, and, optionally, a description.

      A rule can't have the same name as another rule in the same Region and on the same event bus.
   b. For **Event bus**, choose **default**. When an AWS service in your account generates an event, it always goes to your account's default event bus.
   c. For **Rule type**, choose **Rule with an event pattern**.
   d. Choose **Next**.
5. For **Build event pattern**, do the following:
   a. For **Event source**, choose **AWS events or EventBridge partner events**.
   b. For **Event pattern**, do the following:
      i. For **Event source**, choose **AWS services**.
      ii. For **AWS service**, choose **Auto Scaling**.
      iii. For **Event type**, choose **Instance Refresh**.
      iv. By default, the rule matches any instance refresh event. To create a rule that notifies you when a checkpoint is reached during an instance refresh, choose **Specific instance event(s)** and select **EC2 Auto Scaling Instance Refresh Checkpoint Reached**.
      v. By default, the rule matches any Auto Scaling group in the Region. To make the rule match a specific Auto Scaling group, choose **Specific group name(s)** and select one or more Auto Scaling groups.
      vi. Choose **Next**.
6. For **Select target(s)**, do the following:
   a. For **Target types**, choose **AWS service**.
   b. For **Select a target**, choose **SNS topic**.
   c. For **Topic**, choose your Amazon SNS topic.
   d. (Optional) Under **Additional settings**, you can optionally configure additional settings. For more information, see Creating Amazon EventBridge rules that react to events in the *Amazon EventBridge User Guide*.

   e. Choose **Next**.
7. (Optional) For **Tags**, you can optionally assign one or more tags to your rule, and then choose **Next**.
8. For **Review and create**, review the details of the rule and modify them as necessary. Then, choose **Create rule**.

Create EventBridge rules for warm pool events

The following example creates an EventBridge rule to invoke programmatic actions. It does this each time that your Auto Scaling group emits an event when a new instance is added to the warm pool.

Before you create the rule, create the AWS Lambda function that you want the rule to use as a target. You must specify this function as the target for the rule. The following procedure provides only the steps for creating the EventBridge rule that acts when new instances enter the warm pool. For an introductory tutorial that shows you how to create a simple Lambda function to invoke when an incoming event matches a rule, see Tutorial: Configure a lifecycle hook that invokes a Lambda function.

For more information about creating and working with warm pools, see Decrease latency for applications with long boot times using warm pools.

To create an event rule that invokes a Lambda function

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. For **Define rule detail**, do the following:

   a. Enter a **Name** for the rule, and, optionally, a description.

      A rule can't have the same name as another rule in the same Region and on the same event bus.

   b. For **Event bus**, choose **default**. When an AWS service in your account generates an event, it always goes to your account's default event bus.

   c. For **Rule type**, choose **Rule with an event pattern**.

   d. Choose **Next**.

5. For **Build event pattern**, do the following:

   a. For **Event source**, choose **AWS events or EventBridge partner events**.

   b. For **Event pattern**, choose **Custom pattern (JSON editor)**, and paste the following pattern into the **Event pattern** box, replacing the text in *italics* with the name of your Auto Scaling group.

```
{
  "source": [ "aws.autoscaling" ],
  "detail-type": [ "EC2 Instance-launch Lifecycle Action" ],
  "detail": {
      "AutoScalingGroupName": [ "my-asg" ],
      "Origin": [ "EC2" ],
      "Destination": [ "WarmPool" ]
   }
}
```

   To create a rule that matches for other events, modify the event pattern. For more information, see Example event patterns.

   c. Choose **Next**.

6. For **Select target(s)**, do the following:

   a. For **Target types**, choose **AWS service**.

   b. For **Select a target**, choose **Lambda function**.

   c. For **Function**, choose the function that you want to send the events to.

   d. (Optional) For **Configure version/alias**, enter version and alias settings for the target Lambda function.

   e. (Optional) For **Additional settings**, enter any additional settings as appropriate for your application. For more information, see Creating Amazon EventBridge rules that react to events in the *Amazon EventBridge User Guide*.

   f. Choose **Next**.

7. (Optional) For **Tags**, you can optionally assign one or more tags to your rule, and then choose **Next**.

8. For **Review and create**, review the details of the rule and modify them as necessary. Then, choose **Create rule**.

Provide network connectivity for your Auto Scaling instances using Amazon VPC

Amazon Virtual Private Cloud (Amazon VPC) is a service that lets you launch AWS resources such as Auto Scaling groups in a logically isolated virtual network that you define.

A subnet in Amazon VPC is a subdivision within an Availability Zone defined by a segment of the IP address range of the VPC. Using subnets, you can group your instances based on your security and operational needs. A subnet resides entirely within the Availability Zone it was created in. You launch Auto Scaling instances within the subnets.

To enable communication between the internet and the instances in your subnets, you must create an internet gateway and attach it to your VPC. An internet gateway enables your resources within the subnets to connect to the internet through the Amazon EC2 network edge. If a subnet's traffic is routed to an internet gateway, the subnet is known as a *public* subnet. If a subnet's traffic is not routed to an internet gateway, the subnet is known as a *private* subnet. Use a public subnet for resources that must be connected to the internet, and a private subnet for resources that need not be connected to the internet. For more information about giving internet access to instances in a VPC, see Access the internet in the *Amazon VPC User Guide*.

Contents

- Default VPC
- Nondefault VPC
- Considerations when choosing VPC subnets
- IP addressing in a VPC
- Network interfaces in a VPC
- Instance placement tenancy
- AWS Outposts
- More resources for learning about VPCs

Default VPC

If you created your AWS account after December 4, 2013 or you are creating your Auto Scaling group in a new AWS Region, we create a default VPC for you. Your default VPC comes with a

default subnet in each Availability Zone. If you have a default VPC, your Auto Scaling group is created in the default VPC by default.

You can view your VPCs on the Your VPCs page of the Amazon VPC console.

For more information about the default VPC, see Default VPCs in the *Amazon VPC User Guide*.

Nondefault VPC

You can choose to create additional VPCs by going to the VPC Dashboard page in the AWS Management Console and selecting **Create VPC**.

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

> **Note**
>
> A VPC spans all Availability Zones in its AWS Region. When you add subnets to your VPC, choose multiple Availability Zones to ensure that the applications hosted in those subnets are highly available. An Availability Zone is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. Availability Zones help you to make production applications highly available, fault tolerant, and scalable.

Considerations when choosing VPC subnets

Note the following considerations when choosing VPC subnets for your Auto Scaling group:

- If you're attaching an Elastic Load Balancing load balancer to your Auto Scaling group, the instances can be launched into either public or private subnets. However, the load balancer must be created in public subnets to support DNS resolution.
- If you're accessing your Auto Scaling instances directly through SSH, the instances can be launched into public subnets only.
- If you're accessing no-ingress Auto Scaling instances using AWS Systems Manager Session Manager, the instances can be launched into either public or private subnets.
- If you're using private subnets, you can allow the Auto Scaling instances to access the internet by using a public NAT gateway.
- By default, the default subnets in a default VPC are public subnets.

IP addressing in a VPC

When you launch your Auto Scaling instances in a VPC, your instances are automatically assigned a private IP address from the CIDR range of the subnet in which the instance is launched. This enables your instances to communicate with other instances in the VPC.

You can configure a launch template or launch configuration to assign public IPv4 addresses to your instances. Assigning public IP addresses to your instances enables them to communicate with the internet or other AWS services.

When you launch instances into a subnet that is configured to automatically assign IPv6 addresses, they receive both IPv4 and IPv6 addresses. Otherwise, they receive only IPv4 addresses. For more information, see IPv6 addresses in the *Amazon EC2 User Guide*.

For information on specifying CIDR ranges for your VPC or subnet, see the Amazon VPC User Guide.

Amazon EC2 Auto Scaling can automatically assign additional private IP addresses on instance launch when you use a launch template that specifies additional network interfaces. Each network interface is assigned a single private IP address from the CIDR range of the subnet in which the instance is launched. In this case, the system can no longer auto-assign a public IPv4 address to the primary network interface. You will not be able to connect to your instances over a public IPv4 address unless you associate available Elastic IP addresses to the Auto Scaling instances.

Network interfaces in a VPC

Each instance in your VPC has a default network interface (the primary network interface). You cannot detach a primary network interface from an instance. You can create and attach an additional network interface to any instance in your VPC. The number of network interfaces you can attach varies by instance type.

When launching an instance using a launch template, you can specify additional network interfaces. However, launching an Auto Scaling instance with multiple network interfaces automatically creates each interface in the same subnet as the instance. This is because Amazon EC2 Auto Scaling ignores the subnets defined in the launch template in favor of what is specified in the Auto Scaling group. For more information, see Creating a launch template for an Auto Scaling group.

If you create or attach two or more network interfaces from the same subnet to an instance, you might encounter networking issues such as asymmetric routing, especially on instances using

a variant of non-Amazon Linux. If you need this type of configuration, you must configure the secondary network interface within the OS. For an example, see How can I make my secondary network interface work in my Ubuntu EC2 instance? in the AWS Knowledge Center.

Instance placement tenancy

By default, all instances in the VPC run as shared tenancy instances. Amazon EC2 Auto Scaling also supports Dedicated Instances and Dedicated Hosts. For more information, see Create a launch template using advanced settings.

AWS Outposts

AWS Outposts extends an Amazon VPC from an AWS Region to an Outpost with the VPC components that are accessible in the Region, including internet gateways, virtual private gateways, Amazon VPC Transit Gateways, and VPC endpoints. An Outpost is homed to an Availability Zone in the Region and is an extension of that Availability Zone that you can use for resiliency.

For more information, see the AWS Outposts User Guide.

For an example of how to deploy an Auto Scaling group that serves traffic from an Application Load Balancer within an Outpost, see the following blog post Configuring an Application Load Balancer on AWS Outposts.

More resources for learning about VPCs

Use the following topics to learn more about VPCs and subnets.

- Private subnets in a VPC
  - Example: VPC with servers in private subnets and NAT
  - NAT gateways
- Public subnets in a VPC
  - Example: VPC for a test environment
  - Example: VPC for web and database servers
- Subnets for your Application Load Balancer
  - Subnets for your load balancer
- General VPC information

- Amazon VPC User Guide
- Connect VPCs using VPC peering
- Elastic network interfaces
- Use VPC endpoints for private connectivity

Security in Amazon EC2 Auto Scaling

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

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

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon EC2 Auto Scaling, see AWS services in scope by compliance program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon EC2 Auto Scaling. The following topics show you how to configure Amazon EC2 Auto Scaling to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EC2 Auto Scaling resources.

Topics

- Infrastructure security in Amazon EC2 Auto Scaling
- Resilience in Amazon EC2 Auto Scaling
- Data protection in Amazon EC2 Auto Scaling
- Identity and Access Management for Amazon EC2 Auto Scaling
- Compliance validation for Amazon EC2 Auto Scaling
- Amazon EC2 Auto Scaling and interface VPC endpoints

Infrastructure security in Amazon EC2 Auto Scaling

As a managed service, Amazon EC2 Auto Scaling is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud

Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon EC2 Auto Scaling through the network. Clients must support the following:

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

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can also use a virtual private cloud (VPC) endpoint for Amazon EC2 Auto Scaling. Interface VPC endpoints enable your Amazon VPC resources to use their private IP addresses to access Amazon EC2 Auto Scaling with no exposure to the public internet. For more information, see Amazon EC2 Auto Scaling and interface VPC endpoints

Related resources

For information on features for isolating service traffic provided by Amazon EC2, see Infrastructure security in Amazon EC2 in the *Amazon EC2 User Guide*.

Resilience in Amazon EC2 Auto Scaling

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

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

To benefit from the geographic redundancy of the Availability Zone design, do the following:

- Span your Auto Scaling group across multiple Availability Zones.

- Maintain at least one instance in each Availability Zone.
- Attach a load balancer to distribute incoming traffic across the same Availability Zones. If you use an Application Load Balancer, make sure that each EC2 instance gets a similar amount of traffic by keeping cross-zone load balancing enabled. This helps limit the impact of increased load on existing instances during a failover event and results in greater resiliency than without cross-zone load balancing.
- Make sure that the Elastic Load Balancing health checks are configured correctly, and also that they are enabled on the Auto Scaling group. Then, if an instance fails its health check, Elastic Load Balancing stops sending traffic to it and reroutes traffic to healthy instances, while Amazon EC2 Auto Scaling replaces the unhealthy instance.

Amazon EC2 Auto Scaling helps support your application resiliency needs in the following ways:

- Checks instances for health and reachability issues. When an instance becomes unhealthy, it automatically terminates the instance and launches a new one.
- If dynamic scaling policies are in effect, automatically scales capacity according to incoming traffic.
- Detects issues in the reliability of the Amazon CloudWatch metrics that support scaling policies and pauses scale-in activities when reliable metrics are not available, such as when data points are missing.
- Tries to maintain equivalent numbers of instances in each enabled Availability Zone as your group scales.
- Uses Availability Zones to maintain high availability. When an Availability Zone becomes unhealthy, Amazon EC2 Auto Scaling does the following:
  - Launches new instances in a different Availability Zone that is enabled for your Auto Scaling group.
  - Redistributes instances across all enabled Availability Zones when the unhealthy Availability Zone returns to a healthy state.
- Keeps trying to launch instances in other enabled Availability Zones if an instance fails to launch in a given Availability Zone.
- Automatically registers and deregisters instances with the load balancers associated with your Auto Scaling group. This way, you don't need to register and deregister instances separately.
- Control plane outages for the Amazon EC2 Auto Scaling service APIs will not affect the scaling of existing Auto Scaling Groups.

Related resources

For information on features to help support your data resiliency needs provided by Amazon EBS, see Resilience in Amazon Elastic Block Store in the *Amazon EBS User Guide*.

Data protection in Amazon EC2 Auto Scaling

The AWS shared responsibility model applies to data protection in Amazon EC2 Auto Scaling. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

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

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

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon EC2 Auto Scaling or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names

may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

When you launch an Amazon EC2 instance, you have the option of passing user data to the instance to do additional configuration when the instance boots. We also recommend that you never put confidential or sensitive information in the user data that will get passed to an instance.

Use AWS KMS keys to encrypt Amazon EBS volumes

You can configure your Auto Scaling group to encrypt Amazon EBS volume data stored in the cloud with AWS KMS keys. Amazon EC2 Auto Scaling supports AWS managed and customer managed keys to encrypt your data. Note that the `KmsKeyId` option to specify a customer managed key is not available when you use a launch configuration. To specify your customer managed key, use a launch template instead. For more information, see Create a launch template for an Auto Scaling group. For information about how to create, store, and manage your AWS KMS encryption keys, see the AWS Key Management Service Developer Guide.

You can also configure a customer managed key in your EBS-backed AMI before setting up the launch template or launch configuration, or use encryption by default to enforce the encryption of the new EBS volumes and snapshot copies that you create. For more information, see Use encryption with EBS-backed AMIs in the *Amazon EC2 User Guide* and Encryption by default in the *Amazon EBS User Guide*.

> **Note**
>
> For information about how to set up the key policy that you need to launch Auto Scaling instances when you use a customer managed key for encryption, see Required AWS KMS key policy for use with encrypted volumes.

Related resources

For the data protection guidelines provided by Amazon EBS, see Data protection in Amazon Elastic Block Store in the *Amazon EBS User Guide*.

Required AWS KMS key policy for use with encrypted volumes

Amazon EC2 Auto Scaling uses service-linked roles to delegate permissions to other AWS services. Amazon EC2 Auto Scaling service-linked roles are predefined and include permissions that Amazon EC2 Auto Scaling requires to call other AWS services on your behalf. The predefined permissions also include access to your AWS managed keys. However, they do not include access to your customer managed keys, allowing you to maintain full control over these keys.

This topic describes how to set up the key policy that you need to launch Auto Scaling instances when you specify a customer managed key for Amazon EBS encryption.

> **Note**
>
> Amazon EC2 Auto Scaling does not need additional authorization to use the default AWS managed key to protect the encrypted volumes in your account.

Contents

- Overview
- Configure key policies
- Example 1: Key policy sections that allow access to the customer managed key
- Example 2: Key policy sections that allow cross-account access to the customer managed key
- Edit key policies in the AWS KMS console

Overview

The following AWS KMS keys can be used for Amazon EBS encryption when Amazon EC2 Auto Scaling launches instances:

- AWS managed key – An encryption key in your account that Amazon EBS creates, owns, and manages. This is the default encryption key for a new account. The AWS managed key is used for encryption unless you specify a customer managed key.
- Customer managed key – A custom encryption key that you create, own, and manage. For more information, see Creating keys in the *AWS Key Management Service Developer Guide*.

  Note: The key must be symmetric. Amazon EBS does not support asymmetric customer managed keys.

You configure customer managed keys when creating encrypted snapshots or a launch template that specifies encrypted volumes, or enabling encryption by default.

Configure key policies

Your KMS keys must have a key policy that allows Amazon EC2 Auto Scaling to launch instances with Amazon EBS volumes encrypted with a customer managed key.

Use the examples on this page to configure a key policy to give Amazon EC2 Auto Scaling access to your customer managed key. You can modify the customer managed key's key policy either when the key is created or at a later time.

You must, at minimum, add two policy statements to your key policy for it to work with Amazon EC2 Auto Scaling.

- The first statement allows the IAM identity specified in the `Principal` element to use the customer managed key directly. It includes permissions to perform the AWS KMS `Encrypt`, `Decrypt`, `ReEncrypt*`, `GenerateDataKey*`, and `DescribeKey` operations on the key.
- The second statement allows the IAM identity specified in the `Principal` element to use the `CreateGrant` operation to generate grants that delegate a subset of its own permissions to AWS services that are integrated with AWS KMS or another principal. This allows them to use the key to create encrypted resources on your behalf.

When you add the new policy statements to your key policy, do not change any existing statements in the policy.

For each of the following examples, arguments that must be replaced, such as a key ID or the name of a service-linked role, are shown as *`user placeholder text`*. In most cases, you can replace the name of the service-linked role with the name of an Amazon EC2 Auto Scaling service-linked role.

For more information, see the following resources:

- To create a key with the AWS CLI, see create-key.
- To update a key policy with the AWS CLI, see put-key-policy.
- To find a key ID and Amazon Resource Name (ARN), see Finding the key ID and ARN in the *AWS Key Management Service Developer Guide*.
- For information about Amazon EC2 Auto Scaling service-linked roles, see Service-linked roles for Amazon EC2 Auto Scaling.

- For information about Amazon EBS encryption and KMS generally, Amazon EBS encryption in the *Amazon EBS User Guide* and the AWS Key Management Service Developer Guide.

Example 1: Key policy sections that allow access to the customer managed key

Add the following two policy statements to the key policy of the customer managed key, replacing the example ARN with the ARN of the appropriate service-linked role that is allowed access to the key. In this example, the policy sections give the service-linked role named **AWSServiceRoleForAutoScaling** permissions to use the customer managed key.

```
{
   "Sid": "Allow service-linked role use of the customer managed key",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
           "arn:aws:iam::account-id:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"
       ]
   },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
       "kms:DescribeKey"
   ],
   "Resource": "*"
}
```

```
{
   "Sid": "Allow attachment of persistent resources",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
           "arn:aws:iam::account-id:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"
       ]
   },
   "Action": [
       "kms:CreateGrant"
   ],
```

```
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": true
        }
     }
}
```

Example 2: Key policy sections that allow cross-account access to the customer managed key

If you create a customer managed key in a different account than the Auto Scaling group, you must use a grant in combination with the key policy to allow cross-account access to the key.

There are two steps that must be completed in the following order:

1. First, add the following two policy statements to the customer managed key's key policy. Replace the example ARN with the ARN of the other account, making sure to replace *111122223333* with the actual account ID of the AWS account that you want to create the Auto Scaling group in. This allows you to give an IAM user or role in the specified account permission to create a grant for the key using the CLI command that follows. However, this does not by itself give any users access to the key.

```
{
   "Sid": "Allow external account 111122223333 use of the customer managed key",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
           "arn:aws:iam::111122223333:root"
       ]
   },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
       "kms:DescribeKey"
   ],
   "Resource": "*"
}
```

```
{
   "Sid": "Allow attachment of persistent resources in external
 account 111122223333",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
           "arn:aws:iam::111122223333:root"
       ]
   },
   "Action": [
       "kms:CreateGrant"
   ],
   "Resource": "*"
}
```

2. Then, from the account that you want to create the Auto Scaling group in, create a grant that delegates the relevant permissions to the appropriate service-linked role. The `Grantee Principal` element of the grant is the ARN of the appropriate service-linked role. The `key-id` is the ARN of the key.

   The following is an example create-grant CLI command that gives the service-linked role named **AWSServiceRoleForAutoScaling** in account *111122223333* permissions to use the customer managed key in account *444455556666*.

```
aws kms create-grant \
  --region us-west-2 \
  --key-id arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d \
  --grantee-principal arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling \
  --operations "Encrypt" "Decrypt" "ReEncryptFrom" "ReEncryptTo" "GenerateDataKey"
 "GenerateDataKeyWithoutPlaintext" "DescribeKey" "CreateGrant"
```

   For this command to succeed, the user making the request must have permissions for the `CreateGrant` action.

   The following example IAM policy allows an IAM identity (user or role) in account *111122223333* to create a grant for the customer managed key in account *444455556666*.

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreationOfGrantForTheKMSKeyinExternalAccount444455556666",
      "Effect": "Allow",
      "Action": "kms:CreateGrant",
      "Resource": "arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    }
  ]
}
```

For more information about creating a grant for a KMS key in a different AWS account, see Grants in AWS KMS in the *AWS Key Management Service Developer Guide*.

> **Important**
>
> The service-linked role name specified as the grantee principal must be the name of an existing role. After creating the grant, to ensure that the grant allows Amazon EC2 Auto Scaling to use the specified KMS key, do not delete and recreate the service-linked role.

Edit key policies in the AWS KMS console

The examples in the previous sections show only how to add statements to a key policy, which is just one way of changing a key policy. The easiest way to change a key policy is to use the AWS KMS console's default view for key policies and make an IAM identity (user or role) one of the *key users* for the appropriate key policy. For more information, see Using the AWS Management Console default view in the *AWS Key Management Service Developer Guide*.

> **Important**
>
> Be cautious. The console's default view policy statements include permissions to perform AWS KMS `Revoke` operations on the customer managed key. If you give an AWS account access to a customer managed key in your account, and you accidentally revoke the grant that gave them this permission, external users can no longer access their encrypted data or the key that was used to encrypt their data.

Identity and Access Management for Amazon EC2 Auto Scaling

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon EC2 Auto Scaling resources. IAM is an AWS service that you can use with no additional charge.

To use Amazon EC2 Auto Scaling, you need an AWS account and your security credentials for signing into your account. For more information, see AWS security credentials in the *IAM User Guide*.

For complete IAM documentation, see the IAM User Guide.

Access control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon EC2 Auto Scaling resources. For example, you must have permissions to create Auto Scaling groups, launch instances with launch templates, and so on.

The following sections provide details on how an IAM administrator can use IAM to help secure your Amazon EC2 Auto Scaling resources, by controlling who can perform Amazon EC2 Auto Scaling actions.

We recommend that you read the Amazon EC2 topics first. See Identity and access management for Amazon EC2 in the *Amazon EC2 User Guide*. After reading the topics in this section, you should have a good idea what access control permissions Amazon EC2 offers and how they can fit in with your Amazon EC2 Auto Scaling resource permissions.

Topics

- How Amazon EC2 Auto Scaling works with IAM
- Amazon EC2 Auto Scaling API permissions
- AWS managed policies for Amazon EC2 Auto Scaling
- Service-linked roles for Amazon EC2 Auto Scaling
- Amazon EC2 Auto Scaling identity-based policy examples
- Cross-service confused deputy prevention
- Control Amazon EC2 launch template usage in Auto Scaling groups

- IAM role for applications that run on Amazon EC2 instances

How Amazon EC2 Auto Scaling works with IAM

Before you use IAM to manage access to Amazon EC2 Auto Scaling, learn what IAM features are available to use with Amazon EC2 Auto Scaling.

IAM features you can use with Amazon EC2 Auto Scaling

IAM feature	Amazon EC2 Auto Scaling support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how Amazon EC2 Auto Scaling and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon EC2 Auto Scaling

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can

perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

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

Resource-based policies within Amazon EC2 Auto Scaling

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon EC2 Auto Scaling

Supports policy actions: Yes

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

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API

operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon EC2 Auto Scaling actions, see Actions defined by Amazon EC2 Auto Scaling in the *Service Authorization Reference*.

Policy actions in Amazon EC2 Auto Scaling use the following prefix before the action:

```
autoscaling
```

To specify multiple actions in a single statement, separate them with commas.

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

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

```
"Action": "autoscaling:Describe*"
```

Policy resources for Amazon EC2 Auto Scaling

Supports policy resources: Yes

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

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

You can use ARNs to identify the Auto Scaling groups and launch configurations that the IAM policy applies to.

An Auto Scaling group has the following ARN.

```
"Resource": "arn:aws:autoscaling:region:account-
id:autoScalingGroup:uuid:autoScalingGroupName/asg-name"
```

A launch configuration has the following ARN.

```
"Resource": "arn:aws:autoscaling:region:account-
id:launchConfiguration:uuid:launchConfigurationName/lc-name"
```

To specify an Auto Scaling group with the `CreateAutoScalingGroup` action, you must replace the UUID with a wildcard (*) as shown in the following example.

```
"Resource": "arn:aws:autoscaling:region:account-
id:autoScalingGroup:*:autoScalingGroupName/asg-name"
```

To specify a launch configuration with the `CreateLaunchConfiguration` action, you must replace the UUID with a wildcard (*) as shown in the following example.

```
"Resource": "arn:aws:autoscaling:region:account-
id:launchConfiguration:*:launchConfigurationName/lc-name"
```

For more information about Amazon EC2 Auto Scaling resource types and their ARNs, see Resources defined by Amazon EC2 Auto Scaling in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon EC2 Auto Scaling.

> **Note**
>
> For an example of an IAM policy that uses ARNs to control access to Auto Scaling groups, see Control which Auto Scaling groups can be deleted.

Not all Amazon EC2 Auto Scaling actions support resource-level permissions. For actions that don't support resource-level permissions, you must use a wildcard (*) as the resource.

The following Amazon EC2 Auto Scaling actions do not support resource-level permissions.

- `DescribeAccountLimits`
- `DescribeAdjustmentTypes`
- `DescribeAutoScalingGroups`
- `DescribeAutoScalingInstances`
- `DescribeAutoScalingNotificationTypes`
- `DescribeInstanceRefreshes`
- `DescribeLaunchConfigurations`
- `DescribeLifecycleHooks`
- `DescribeLifecycleHookTypes`
- `DescribeLoadBalancers`
- `DescribeLoadBalancerTargetGroups`
- `DescribeMetricCollectionTypes`
- `DescribeNotificationConfigurations`
- `DescribePolicies`
- `DescribeScalingActivities`
- `DescribeScalingProcessTypes`
- `DescribeScheduledActions`
- `DescribeTags`
- `DescribeTerminationPolicyTypes`
- `DescribeWarmPool`

Policy condition keys for Amazon EC2 Auto Scaling

Supports service-specific policy condition keys: Yes

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

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use

condition operators, such as equals or less than, to match the condition in the policy with values in the request.

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

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

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Amazon EC2 Auto Scaling supports the following condition keys that can be used to control access to supported actions and enforce the configuration of Auto Scaling groups:

- `autoscaling:InstanceTypes`
- `autoscaling:LaunchConfigurationName`
- `autoscaling:LaunchTemplateVersionSpecified`
- `autoscaling:LoadBalancerNames`
- `autoscaling:MaxSize`
- `autoscaling:MinSize`
- `autoscaling:ResourceTag/`*`key-name`*`:` *`tag-value`*
- `autoscaling:TargetGroupARNs`
- `autoscaling:VPCZoneIdentifiers`

The following condition keys are specific to create launch configuration requests:

- `autoscaling:ImageId`
- `autoscaling:InstanceType`
- `autoscaling:MetadataHttpEndpoint`
- `autoscaling:MetadataHttpPutResponseHopLimit`
- `autoscaling:MetadataHttpTokens`
- `autoscaling:SpotPrice`

Amazon EC2 Auto Scaling also supports the following global condition keys that you can use to define permissions based on the tags in the request or present on the Auto Scaling group. For more information, see Tag Auto Scaling groups and instances.

- `aws:RequestTag/key-name: tag-value`
- `aws:ResourceTag/key-name: tag-value`
- `aws:TagKeys: [tag-key, ...]`

To learn which Amazon EC2 Auto Scaling API actions you can use a condition key with, see Actions defined by Amazon EC2 Auto Scaling in the *Service Authorization Reference*. For more information about Amazon EC2 Auto Scaling condition keys, see Condition keys for Amazon EC2 Auto Scaling.

> **Note**
>
> For examples of IAM policies that use condition keys to control access to supported actions and enforce the configuration of Auto Scaling groups, see the following resources:
>
> - Require a launch template and a version number – This example enforces that a launch template and the version number of the launch template must be specified when creating or updating Auto Scaling groups.
> - Control the size of the Auto Scaling groups that can be created – This example enforces constraints on the possible values for the `MinSize` and `MaxSize` properties when creating or updating Auto Scaling groups with a specific tag.
> - Control which scaling policies can be deleted – This example enforces that deleting scaling policies is allowed only for Auto Scaling groups without a specific tag.

ACLs in Amazon EC2 Auto Scaling

Supports ACLs: No

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

ABAC with Amazon EC2 Auto Scaling

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

ABAC is possible for resources that support tags, but not everything supports tags. Launch configurations and scaling policies don't support tags, but Auto Scaling groups support tags.

For more information, see Tag Auto Scaling groups and instances.

Using temporary credentials with Amazon EC2 Auto Scaling

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Service roles for Amazon EC2 Auto Scaling

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

When you create a lifecycle hook that notifies an Amazon SNS topic or Amazon SQS queue, you must specify a role to allow Amazon EC2 Auto Scaling to access Amazon SNS or Amazon SQS on your behalf. Use the IAM console to set up the service role for your lifecycle hook. The console helps you create a role with a sufficient set of permissions using a managed policy. For more information, see Receive notifications using Amazon SNS and Receive notifications using Amazon SQS.

When you create an Auto Scaling group, you can optionally pass in a service role to allow Amazon EC2 instances to access other AWS services on your behalf. The service role for Amazon EC2 instances (also called the Amazon EC2 instance profile for a launch template or launch configuration) is a special type of service role that is assigned to every EC2 instance in an Auto Scaling group when the instance launches. You can use the IAM console and AWS CLI to create or edit this service role. For more information, see IAM role for applications that run on Amazon EC2 instances.

> **Warning**
>
> Changing the permissions for a service role might break Amazon EC2 Auto Scaling functionality. Edit service roles only when Amazon EC2 Auto Scaling provides guidance to do so.

Service-linked roles for Amazon EC2 Auto Scaling

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing Amazon EC2 Auto Scaling service-linked roles, see Service-linked roles for Amazon EC2 Auto Scaling.

Amazon EC2 Auto Scaling API permissions

You must grant users permission to call the Amazon EC2 Auto Scaling API actions they need, as described in Policy actions for Amazon EC2 Auto Scaling. In addition, for some Amazon EC2 Auto Scaling actions, you must grant users permission to call specific actions from other AWS APIs.

Required permissions from other AWS APIs

In addition to Amazon EC2 Auto Scaling API permissions, users must have the following permissions from other AWS APIs to successfully perform the associated action.

Create an Auto Scaling group (`autoscaling:CreateAutoScalingGroup`)

- `iam:CreateServiceLinkedRole` – To create the default service-linked role if that role does not yet exist.
- `iam:PassRole` – To pass an IAM role to the service or to EC2 instances on launch. Needed when a nondefault service-linked role, an IAM role for a lifecycle hook, or a launch template that specifies an instance profile (a container for an IAM role) is provided.
- `ec2:RunInstances` – To launch instances when a launch template is provided.
- `ec2:CreateTags` – To tag instances and volumes on launch when a launch template with a tag specification is provided.

Create a lifecycle hook (`autoscaling:PutLifecycleHook`)

- `iam:PassRole` – To pass an IAM role to the service. Needed when an IAM role is provided.

Attach a VPC Lattice target group (`autoscaling:AttachTrafficSources`)

- `vpc-lattice:RegisterTargets` – To automatically register instances with the target group.

Detach a VPC Lattice target group (`autoscaling:DetachTrafficSources`)

- `vpc-lattice:DeregisterTargets` – To automatically deregister instances with the target group.

Create a launch configuration (`autoscaling:CreateLaunchConfiguration`)

- `ec2:DescribeImages`
- `ec2:DescribeInstances`
- `ec2:DescribeInstanceAttribute`
- `ec2:DescribeKeyPairs`
- `ec2:DescribeSecurityGroups`
- `ec2:DescribeSpotInstanceRequests`
- `ec2:DescribeVpcClassicLink`
- `iam:PassRole` – To pass an IAM role to EC2 instances on launch. Needed when a launch configuration specifies an instance profile (a container for an IAM role).

AWS managed policies for Amazon EC2 Auto Scaling

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

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

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

For more information, see AWS managed policies in the *IAM User Guide*.

Amazon EC2 Auto Scaling managed policies

You can attach the following managed policies to your AWS Identity and Access Management (IAM) identities (users or roles). Each policy provides access to all or some of the API actions for Amazon EC2 Auto Scaling.

- AutoScalingConsoleFullAccess – Grants full access to Amazon EC2 Auto Scaling using the AWS Management Console. This policy works when you are using launch configurations, but not when you are using launch templates.
- AutoScalingConsoleReadOnlyAccess – Grants read-only access to Amazon EC2 Auto Scaling using the AWS Management Console. This policy works when you are using launch configurations, but not when you are using launch templates.
- AutoScalingFullAccess – Grants full access to Amazon EC2 Auto Scaling for IAM identities that need full Amazon EC2 Auto Scaling access from the AWS CLI or SDKs, but not AWS Management Console access.
- AutoScalingReadOnlyAccess – Grants read-only access to Amazon EC2 Auto Scaling for IAM identities that are making calls only to the AWS CLI or SDKs.

When you are using launch templates from the console, you need to grant additional permissions specific to launch templates, which are discussed in Control Amazon EC2 launch template usage in Auto Scaling groups. The Amazon EC2 Auto Scaling console needs permissions for `ec2` actions so it can display information about launch templates and launch instances using launch templates.

AutoScalingServiceRolePolicy AWS managed policy

This policy is attached to a service-linked role that allows Amazon EC2 Auto Scaling to perform actions on your behalf. For more information, see Service-linked roles for Amazon EC2 Auto Scaling.

To view the permissions for this policy, see AutoScalingServiceRolePolicy in the *AWS Managed Policy Reference*.

Amazon EC2 Auto Scaling updates to AWS managed policies

View details about updates to AWS managed policies for Amazon EC2 Auto Scaling since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon EC2 Auto Scaling Document history page.

Change	Description	Date
Amazon EC2 Auto Scaling adds permissions to its service-linked role	The `AutoScalingServiceRolePolicy` policy now grants permissions to call	February 29, 2024

Change	Description	Date
	the Amazon EC2 GetSecurityGroupsForVpc API action to get all security groups for a VPC to improve validation, and the Amazon EC2 GetInstanceTypesFromInstanceRequirements API action to get information about which instance types meet a certain set of instance requirements. For more information, see Service-linked roles for Amazon EC2 Auto Scaling.	

Change	Description	Date
Amazon EC2 Auto Scaling adds permissions to its service-linked role	The `AutoScalingServiceRolePolicy` policy now grants permissions to the service to access the API actions it needs for an integration with VPC Lattice. • `GetTargetGroup` and `ListTargetGroup` actions. Required to retrieve information about VPC Lattice target groups. • `RegisterTargets` and `DeregisterTargets` actions. Required to register and deregister instances from VPC Lattice target groups. • `ListTargets` . Allows Amazon EC2 Auto Scaling to retrieve health informati on for instances registere d to VPC Lattice target groups. For more information, see Service-linked roles for Amazon EC2 Auto Scaling.	December 6, 2022

Change	Description	Date
Amazon EC2 Auto Scaling adds permissions to its service-linked role	To support using an AWS Systems Manager Parameter as an alias for an AMI ID when creating a launch template, the `AutoScalingServiceRolePolicy` policy now grants permission to call the AWS Systems Manager GetParameters API action. For more information, see Service-linked roles for Amazon EC2 Auto Scaling.	March 28, 2022
Amazon EC2 Auto Scaling adds permissions to its service-linked role	To support predictive scaling, the `AutoScalingServiceRolePolicy` policy now includes permission to call the CloudWatch GetMetricData API action. For more information, see Service-linked roles for Amazon EC2 Auto Scaling.	May 19, 2021
Amazon EC2 Auto Scaling started tracking changes	Amazon EC2 Auto Scaling started tracking changes for its AWS managed policies.	May 19, 2021

Service-linked roles for Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling uses service-linked roles for the permissions that it requires to call other AWS services on your behalf. A service-linked role is a unique type of IAM role that is linked directly to an AWS service.

Service-linked roles provide a secure way to delegate permissions to other AWS services because only the linked service can assume a service-linked role. For more information, see Using service-

linked roles in the *IAM User Guide*. Service-linked roles also enable all API calls to be visible through AWS CloudTrail. This helps with monitoring and auditing requirements because you can track all actions that Amazon EC2 Auto Scaling performs on your behalf. For more information, see Log Amazon EC2 Auto Scaling API calls with AWS CloudTrail.

The following sections describe how to create and manage Amazon EC2 Auto Scaling service-linked roles. Start by configuring permissions to allow an IAM identity (such as a user or role) to create, edit, or delete a service-linked role. For more information, see Using service-linked roles in the *IAM User Guide*.

Contents

- Overview
- Permissions granted by the service-linked role
- Supported Regions for Amazon EC2 Auto Scaling service-linked roles
- Create, edit, and delete a service linked role
  - Create a service-linked role (automatic)
  - Create a service-linked role (manual)
  - Edit the service-linked role
  - Delete the service-linked role

Overview

There are two types of Amazon EC2 Auto Scaling service-linked roles:

- The default service-linked role for your account, named **AWSServiceRoleForAutoScaling**. This role is automatically assigned to your Auto Scaling groups unless you specify a different service-linked role.
- A service-linked role with a custom suffix that you specify when you create the role, for example, **AWSServiceRoleForAutoScaling_*mysuffix***.

The permissions of a custom suffix service-linked role are identical to those of the default service-linked role. In both cases, you cannot edit the roles, and you also cannot delete them if they are still in use by an Auto Scaling group. The only difference is the role name suffix.

You can specify either role when you edit your AWS Key Management Service key policies to allow instances that are launched by Amazon EC2 Auto Scaling to be encrypted with your customer

managed key. However, if you plan to give granular access to a specific customer managed key, you should use a custom suffix service-linked role. Using a custom suffix service-linked role provides you with:

- More control over the customer managed key
- The ability to track which Auto Scaling group made an API call in your CloudTrail logs

If you create customer managed keys that not all users should have access to, follow these steps to allow the use of a custom suffix service-linked role:

1. Create a service-linked role with a custom suffix. For more information, see Create a service-linked role (manual).
2. Give the service-linked role access to a customer managed key. For more information about the key policy that allows the key to be used by a service-linked role, see Required AWS KMS key policy for use with encrypted volumes.
3. Give users access to the service-linked role that you created. For more information about creating the IAM policy, see Control which service-linked role can be passed (using PassRole). If users try to specify a service-linked role without permission to pass that role to the service, they receive an error.

Permissions granted by the service-linked role

Amazon EC2 Auto Scaling uses the service-linked role named **AWSServiceRoleForAutoScaling** or your custom suffix service-linked role.

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

- `autoscaling.amazonaws.com`

The role permissions policy, AutoScalingServiceRolePolicy, allows Amazon EC2 Auto Scaling to complete the following actions:

- `ec2` – Create, describe, modify, start/stop, and terminate EC2 instances.
- `iam` – Pass IAM roles to EC2 instances so that applications running on the instances can access temporary credentials for the role.
- `iam` – Create the **AWSServiceRoleForEC2Spot** service-linked role to allow Amazon EC2 Auto Scaling to launch Spot Instances on your behalf.

- `elasticloadbalancing` – Register and deregister instances with Elastic Load Balancing and check the health of registered targets.
- `cloudwatch` – Create, describe, modify, and delete CloudWatch alarms for scaling policies and retrieve metrics used for predictive scaling.
- `sns` – Publish notifications to Amazon SNS when instances launch or terminate.
- `events` – Create, describe, update, and delete EventBridge rules on your behalf.
- `ssm` – Read parameters from Parameter Store when using a Systems Manager parameter as an alias for an AMI ID in a launch template.
- `vpc-lattice` – Register and deregister instances with VPC Lattice and check the health of registered targets.

Supported Regions for Amazon EC2 Auto Scaling service-linked roles

Amazon EC2 Auto Scaling supports using service-linked roles in all of the AWS Regions where the service is available.

Create, edit, and delete a service linked role

Create a service-linked role (automatic)

Amazon EC2 Auto Scaling creates the **AWSServiceRoleForAutoScaling** service-linked role for you the first time that you create an Auto Scaling group, unless you manually create a custom suffix service-linked role and specify it when creating the group.

> **Important**
>
> You must have IAM permissions to create the service-linked role. Otherwise, the automatic creation fails. For more information, see Service-linked role permissions in the *IAM User Guide* and Create a service-linked role in this guide.

Amazon EC2 Auto Scaling began supporting service-linked roles in March 2018. If you created an Auto Scaling group before then, Amazon EC2 Auto Scaling created the **AWSServiceRoleForAutoScaling** role in your account. For more information, see A new role appeared in my AWS account in the *IAM User Guide*.

Create a service-linked role (manual)

To create a service-linked role (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select trusted entity**, choose **AWS service**.
4. For **Choose the service that will use this role**, choose **EC2 Auto Scaling** and the **EC2 Auto Scaling** use case.
5. Choose **Next: Permissions**, **Next: Tags**, and then **Next: Review**. Note: You cannot attach tags to service-linked roles during creation.
6. On the **Review** page, leave **Role name** blank to create a service-linked role with the name **AWSServiceRoleForAutoScaling**, or enter a suffix to create a service-linked role with the name **AWSServiceRoleForAutoScaling_*suffix***.
7. (Optional) For **Role description**, edit the description for the service-linked role.
8. Choose **Create role**.

To create a service-linked role (AWS CLI)

Use the following create-service-linked-role CLI command to create a service-linked role for Amazon EC2 Auto Scaling with the name **AWSServiceRoleForAutoScaling_*suffix***.

```
aws iam create-service-linked-role --aws-service-name autoscaling.amazonaws.com --
custom-suffix suffix
```

The output of this command includes the ARN of the service-linked role, which you can use to give the service-linked role access to your customer managed key.

```
{
    "Role": {
        "RoleId": "ABCDEF0123456789ABCDEF",
        "CreateDate": "2018-08-30T21:59:18Z",
        "RoleName": "AWSServiceRoleForAutoScaling_suffix",
        "Arn": "arn:aws:iam::123456789012:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling_suffix",
        "Path": "/aws-service-role/autoscaling.amazonaws.com/",
        "AssumeRolePolicyDocument": {
```

```
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": [
                        "sts:AssumeRole"
                    ],
                    "Principal": {
                        "Service": [
                            "autoscaling.amazonaws.com"
                        ]
                    },
                    "Effect": "Allow"
                }
            ]
        }
    }
}
```

For more information, see Creating a service-linked role in the *IAM User Guide*.

Edit the service-linked role

You cannot edit the service-linked roles that are created for Amazon EC2 Auto Scaling. After you create a service-linked role, you cannot change the name of the role or its permissions. However, you can edit the description of the role. For more information, see Editing a service-linked role in the *IAM User Guide*.

Delete the service-linked role

If you are not using an Auto Scaling group, we recommend that you delete its service-linked role. Deleting the role prevents you from having an entity that is not used or actively monitored and maintained.

You can delete a service-linked role only after first deleting the related dependent resources. This protects you from inadvertently revoking Amazon EC2 Auto Scaling permissions to your resources. If a service-linked role is used with multiple Auto Scaling groups, you must delete all Auto Scaling groups that use the service-linked role before you can delete it. For more information, see Delete your Auto Scaling infrastructure.

You can use IAM to delete a service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

If you delete the **AWSServiceRoleForAutoScaling** service-linked role, Amazon EC2 Auto Scaling creates the role again when you create an Auto Scaling group and do not specify a different service-linked role.

Amazon EC2 Auto Scaling identity-based policy examples

By default, a brand new user in your AWS account has no permissions to do anything. An IAM administrator must create and assign IAM policies that give an IAM identity (such as a user or role) permission to perform Amazon EC2 Auto Scaling API actions.

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

The following shows an example of a permissions policy.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "autoscaling:CreateAutoScalingGroup",
          "autoscaling:UpdateAutoScalingGroup",
          "autoscaling:DeleteAutoScalingGroup"
      ],
      "Resource": "*",
      "Condition": {
          "StringEquals": { "autoscaling:ResourceTag/purpose": "testing" }
      }
   },
   {
      "Effect": "Allow",
      "Action": "autoscaling:Describe*",
      "Resource": "*"
   }]
}
```

This sample policy grants permissions to create, update, and delete Auto Scaling groups, but only if the group uses the tag **`purpose=testing`**. Because `Describe` actions do not support resource-level permissions, you must specify them in a separate statement without conditions. To launch instances with a launch template, the user must also have the `ec2:RunInstances` permission. For more information, see Control Amazon EC2 launch template usage in Auto Scaling groups.

> **Note**
>
> You can create your own custom IAM policies to allow or deny permissions for IAM identities (users or roles) to perform Amazon EC2 Auto Scaling actions. You can attach these custom policies to the IAM identities that require the specified permissions. The following examples show permissions for some common use cases.
> Some Amazon EC2 Auto Scaling API actions allow you to include specific Auto Scaling groups in your policy that can be created or modified by the action. You can restrict the target resources for these actions by specifying individual Auto Scaling group ARNs. As a best practice, however, we recommend that you use tag-based policies that allow (or deny) actions on Auto Scaling groups with a specific tag.

Examples

- Control the size of the Auto Scaling groups that can be created
- Control which tag keys and tag values can be used
- Control which Auto Scaling groups can be deleted
- Control which scaling policies can be deleted
- Control access to instance refresh actions
- Create a service-linked role
- Control which service-linked role can be passed (using PassRole)

Control the size of the Auto Scaling groups that can be created

The following policy grants permissions to create and update all Auto Scaling groups with the tag **`environment=development`**, as long as the requester doesn't specify a minimum size less than **`1`** or a maximum size greater than **`10`**. Whenever possible, use tags to help you control access to the Auto Scaling groups in your account.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "autoscaling:CreateAutoScalingGroup",
          "autoscaling:UpdateAutoScalingGroup"
```

```
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": { "autoscaling:ResourceTag/environment": "development" },
            "NumericGreaterThanEqualsIfExists": { "autoscaling:MinSize": 1 },
            "NumericLessThanEqualsIfExists": { "autoscaling:MaxSize": 10 }
        }
    }]
}
```

Alternatively, if you are not using tags to control access to Auto Scaling groups, you can use ARNs to identify the Auto Scaling groups that the IAM policy applies to.

An Auto Scaling group has the following ARN.

```
"Resource": "arn:aws:autoscaling:region:account-
id:autoScalingGroup:*:autoScalingGroupName/my-asg"
```

You can also specify multiple ARNs by enclosing them in a list. For more information about specifying the ARNs of Amazon EC2 Auto Scaling resources in the `Resource` element, see Policy resources for Amazon EC2 Auto Scaling.

Control which tag keys and tag values can be used

You can also use conditions in your IAM policies to control the tag keys and tag values that can be applied to Auto Scaling groups. To grant permissions to create or tag an Auto Scaling group only if the requester specifies certain tags, use the `aws:RequestTag` condition key. To allow only specific tag keys, use the `aws:TagKeys` condition key with the `ForAllValues` modifier.

The following policy requires the requester to specify a tag with the key **`environment`** in the request. The **`"?*"`** value enforces that there is some value for the tag key. To use a wildcard, you must use the `StringLike` condition operator.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "autoscaling:CreateAutoScalingGroup",
          "autoscaling:CreateOrUpdateTags"
```

```
      ],
      "Resource": "*",
      "Condition": {
          "StringLike": { "aws:RequestTag/environment": "?*" }
      }
   }]
}
```

The following policy specifies that the requester can only tag Auto Scaling groups with the tags **`purpose=webserver`** and **`cost-center=cc123`**, and allows only the **`purpose`** and **`cost-center`** tags (no other tags can be specified).

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "autoscaling:CreateAutoScalingGroup",
          "autoscaling:CreateOrUpdateTags"
      ],
      "Resource": "*",
      "Condition": {
          "StringEquals": {
              "aws:RequestTag/purpose": "webserver",
              "aws:RequestTag/cost-center": "cc123"
          },
          "ForAllValues:StringEquals": { "aws:TagKeys": ["purpose", "cost-center"] }
      }
   }]
}
```

The following policy requires the requester to specify at least one tag in the request, and allows only the **`cost-center`** and **`owner`** keys.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "autoscaling:CreateAutoScalingGroup",
          "autoscaling:CreateOrUpdateTags"
      ],
```

```
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": { "aws:TagKeys": ["cost-center", "owner"] }
        }
    }]
}
```

> **Note**
>
> For conditions, the condition key is not case-sensitive and the condition value is case-sensitive. Therefore, to enforce the case-sensitivity of a tag key, use the `aws:TagKeys` condition key, where the tag key is specified as a value in the condition.

Control which Auto Scaling groups can be deleted

The following policy allows deletion of an Auto Scaling group only if the group has the tag **`environment=development`**.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "autoscaling:DeleteAutoScalingGroup",
        "Resource": "*",
        "Condition": {
            "StringEquals": { "aws:ResourceTag/environment": "development" }
        }
    }]
}
```

Alternatively, if you are not using condition keys to control access to Auto Scaling groups, you can specify the ARNs of resources in the `Resource` element to control access instead.

The following policy gives users permissions to use the `DeleteAutoScalingGroup` API action, but only for only for Auto Scaling groups whose name begins with **`devteam-`**.

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

```
      "Effect": "Allow",
      "Action": "autoscaling:DeleteAutoScalingGroup",
      "Resource": "arn:aws:autoscaling:region:account-
id:autoScalingGroup:*:autoScalingGroupName/devteam-*"
   }]
}
```

You can also specify multiple ARNs by enclosing them in a list. Including the UUID ensures that access is granted to the specific Auto Scaling group. The UUID for a new group is different than the UUID for a deleted group with the same name.

```
"Resource": [
    "arn:aws:autoscaling:region:account-
id:autoScalingGroup:uuid:autoScalingGroupName/devteam-1",
    "arn:aws:autoscaling:region:account-
id:autoScalingGroup:uuid:autoScalingGroupName/devteam-2",
    "arn:aws:autoscaling:region:account-
id:autoScalingGroup:uuid:autoScalingGroupName/devteam-3"
]
```

Control which scaling policies can be deleted

The following policy grants permissions to use the `DeletePolicy` action to delete a scaling policy. However, it also denies the action if the Auto Scaling group being acted upon has the tag **`environment=production`**. Whenever possible, use tags to help you control access to the Auto Scaling groups in your account.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "autoscaling:DeletePolicy",
      "Resource": "*"
   },
   {
      "Effect": "Deny",
      "Action": "autoscaling:DeletePolicy",
      "Resource": "*",
      "Condition": {
          "StringEquals": { "autoscaling:ResourceTag/environment": "production" }
      }
```

```
    }]
}
```

Control access to instance refresh actions

The following policy grants permission to start, roll back, and cancel an instance refresh only if the Auto Scaling group being acted upon has the tag **`environment=testing`**. Because `Describe` actions do not support resource-level permissions, you must specify them in a separate statement without conditions.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "autoscaling:StartInstanceRefresh",
          "autoscaling:CancelInstanceRefresh",
          "autoscaling:RollbackInstanceRefresh"
      ],
      "Resource": "*",
      "Condition": {
          "StringEquals": { "autoscaling:ResourceTag/environment": "testing" }
      }
   },
   {
      "Effect": "Allow",
      "Action": "autoscaling:DescribeInstanceRefreshes",
      "Resource": "*"
   }]
}
```

To specify a desired configuration in the `StartInstanceRefresh` call, users might need some related permissions, such as:

- **ec2:RunInstances** – To launch EC2 instances using a launch template, the user must have the `ec2:RunInstances` permission in an IAM policy. For more information, see Control Amazon EC2 launch template usage in Auto Scaling groups.
- **ec2:CreateTags** – To launch EC2 instances from a launch template that adds tags to the instances and volumes on creation, the user must have the `ec2:CreateTags` permission in an IAM policy. For more information, see Permissions required to tag instances and volumes.

- **iam:PassRole** – To launch EC2 instances from a launch template that contains an instance profile (a container for an IAM role), the user must also have the `iam:PassRole` permission in an IAM policy. For more information and an example IAM policy, see IAM role for applications that run on Amazon EC2 instances.
- **ssm:GetParameters** – To launch EC2 instances from a launch template that uses an AWS Systems Manager parameter, the user must also have the `ssm:GetParameters` permission in an IAM policy. For more information, see Use AWS Systems Manager parameters instead of AMI IDs in launch templates.

Create a service-linked role

Amazon EC2 Auto Scaling requires permissions to create a service-linked role the first time that any user in your AWS account calls Amazon EC2 Auto Scaling API actions. If the service-linked role does not exist already, Amazon EC2 Auto Scaling creates it in your account. The service-linked role gives permissions to Amazon EC2 Auto Scaling so that it can call other AWS services on your behalf.

For automatic role creation to succeed, users must have permissions for the `iam:CreateServiceLinkedRole` action.

```
"Action": "iam:CreateServiceLinkedRole"
```

The following shows an example of a permissions policy that allows a user to create an Amazon EC2 Auto Scaling service-linked role for Amazon EC2 Auto Scaling.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling",
      "Condition": {
          "StringLike": { "iam:AWSServiceName":"autoscaling.amazonaws.com" }
      }
   }]
}
```

Control which service-linked role can be passed (using PassRole)

Users who create or update Auto Scaling groups and specify a custom suffix service-linked role in the request require the `iam:PassRole` permission.

You can use the `iam:PassRole` permission to protect the security of your AWS KMS customer managed keys if you give different service-linked roles access to different keys. Depending on your organization's needs, you might have a key for the development team, another for the QA team, and another for the finance team. First, create a service-linked role that has access to the required key, for example, a service-linked role named **AWSServiceRoleForAutoScaling_devteamkeyaccess**. Then, attach the policy to an IAM identity, such as a user or role.

The following policy grants permissions to pass the **`AWSServiceRoleForAutoScaling_devteamkeyaccess`** role to any Auto Scaling group whose name begins with **`devteam-`**. If the IAM identity that creates the Auto Scaling group tries to specify a different service-linked role, they receive an error. If they choose not to specify a service-linked role, the default **AWSServiceRoleForAutoScaling** role is used instead.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::account-id:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling_devteamkeyaccess",
      "Condition": {
          "StringEquals": { "iam:PassedToService": [ "autoscaling.amazonaws.com" ] },
          "StringLike": { "iam:AssociatedResourceARN":
 [ "arn:aws:autoscaling:region:account-
id:autoScalingGroup:*:autoScalingGroupName/devteam-*" ] }
      }
   }]
}
```

For more information about custom suffix service-linked roles, see Service-linked roles for Amazon EC2 Auto Scaling.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action.

In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access.

To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account. We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in trust policies for Amazon EC2 Auto Scaling service roles. These keys limit the permissions that Amazon EC2 Auto Scaling gives another service to the resource.

The values for the `SourceArn` and `SourceAccount` fields are set when Amazon EC2 Auto Scaling uses AWS Security Token Service (AWS STS) to assume a role on your behalf.

To use the `aws:SourceArn` or `aws:SourceAccount` global condition keys, set the value to the Amazon Resource Name (ARN) or account of the resource that Amazon EC2 Auto Scaling stores. Whenever possible, use `aws:SourceArn`, which is more specific. Set the value to the ARN or an ARN pattern with wildcards (`*`) for the unknown portions of the ARN. If you don't know the ARN of the resource, use `aws:SourceAccount` instead.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon EC2 Auto Scaling to prevent the confused deputy problem.

Example: Using `aws:SourceArn` and `aws:SourceAccount` condition keys

A role that a service assumes to perform actions on your behalf is called a service role. In cases where you want to create lifecycle hooks that send notifications to anywhere other than Amazon EventBridge, you must create a service role to allow Amazon EC2 Auto Scaling to send notifications to an Amazon SNS topic or Amazon SQS queue on your behalf. If you want only one Auto Scaling group to be associated with the cross-service access, you can specify the trust policy of the service role as follows.

This example trust policy uses condition statements to limit the `AssumeRole` capability on the service role to only the actions that affect the specified Auto Scaling group in the specified account. The `aws:SourceArn` and `aws:SourceAccount` conditions are evaluated independently. Any request to use the service role must satisfy both conditions.

Before using this policy, replace the Region, account ID, UUID, and group name with valid values from your account.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "autoscaling.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn":
 "arn:aws:autoscaling:region:account_id:autoScalingGroup:uuid:autoScalingGroupName/my-
asg"
      },
      "StringEquals": {
        "aws:SourceAccount": "account_id"
      }
    }
  }
}
```

In the preceding example:

- The `Principal` element specifies the service principal of the service (`autoscaling.amazonaws.com`).
- The `Action` element specifies the `sts:AssumeRole` action.
- The `Condition` element specifies the `aws:SourceArn` and `aws:SourceAccount` global condition keys. The source's ARN includes the account ID, so it is not necessary to use `aws:SourceAccount` with `aws:SourceArn`.

Additional information

For more information, see AWS global condition context keys, The confused deputy problem, and Modifying a role trust policy (console) in the *IAM User Guide*.

Control Amazon EC2 launch template usage in Auto Scaling groups

Amazon EC2 Auto Scaling supports using Amazon EC2 launch templates with your Auto Scaling groups. We recommend that you allow users to create Auto Scaling groups from launch templates,

because doing so allows them to use the latest features of Amazon EC2 Auto Scaling and Amazon EC2. For example, users must specify a launch template to use a mixed instances policy.

You can use the `AmazonEC2FullAccess` policy to give users complete access to work with Amazon EC2 Auto Scaling resources, launch templates, and other EC2 resources in their account. Or, you can create your own custom IAM policies to give users fine-grained permissions to work with launch templates, as described in this topic.

A sample policy that you can tailor for your own use

The following shows an example of a basic permissions policy that you can tailor for your own use. The policy grants permissions to create, update, and delete all Auto Scaling groups, but only if the group uses the tag **`purpose=testing`**. It then gives permission for all `Describe` actions. Because `Describe` actions do not support resource-level permissions, you must specify them in a separate statement without conditions.

IAM identities (users or roles) with this policy have permission to create or update an Auto Scaling group using a launch template because they're also given permission to use the `ec2:RunInstances` action.

```
{
    "Version": "2012-10-17",
    "Statement": [
          {
            "Effect": "Allow",
            "Action": [
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:DeleteAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": { "autoscaling:ResourceTag/purpose": "testing" }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:Describe*",
                "ec2:RunInstances"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Users who create or update Auto Scaling groups might need some related permissions, such as:

- **ec2:CreateTags** – To add tags to the instances and volumes on creation, the user must have the `ec2:CreateTags` permission in an IAM policy. For more information, see Permissions required to tag instances and volumes.
- **iam:PassRole** – To launch EC2 instances from a launch template that contains an instance profile (a container for an IAM role), the user must also have the `iam:PassRole` permission in an IAM policy. For more information and an example IAM policy, see IAM role for applications that run on Amazon EC2 instances.
- **ssm:GetParameters** – To launch EC2 instances from a launch template that uses an AWS Systems Manager parameter, the user must also have the `ssm:GetParameters` permission in an IAM policy. For more information, see Use AWS Systems Manager parameters instead of AMI IDs in launch templates.

These permissions for actions to be completed when launching instances are checked when the user interacts with an Auto Scaling group. For more information, see Permissions validation for `ec2:RunInstances` and `iam:PassRole`.

The following examples show policy statements that you could use to control the access that IAM users have to using launch templates.

Topics

- Require launch templates that have a specific tag
- Require a launch template and a version number
- Require the use of instance metadata service version 2 (IMDSv2)
- Restrict access to Amazon EC2 resources
- Permissions required to tag instances and volumes
- Additional launch template permissions
- Permissions validation for ec2:RunInstances and iam:PassRole
- Related resources

Require launch templates that have a specific tag

When granting `ec2:RunInstances` permissions, you can specify that users can only use launch templates with specific tags or specific IDs to limit permissions when launching instances with a launch template. You can also control the AMI and other resources that anyone using launch templates can reference and use when launching instances by specifying additional resource-level permissions for the `RunInstances` call.

The following example restricts permissions for the `ec2:RunInstances` action to launch templates that are located in the specified Region and that have the tag **`purpose=testing`**. It also gives users access to the resources specified in a launch template: AMIs, instance types, volumes, key pairs, network interfaces, and security groups.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": "arn:aws:ec2:region:account-id:launch-template/*",
            "Condition": {
                "StringEquals": { "aws:ResourceTag/purpose": "testing" }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:region::image/ami-*",
                "arn:aws:ec2:region:account-id:instance/*",
                "arn:aws:ec2:region:account-id:subnet/*",
                "arn:aws:ec2:region:account-id:volume/*",
                "arn:aws:ec2:region:account-id:key-pair/*",
                "arn:aws:ec2:region:account-id:network-interface/*",
                "arn:aws:ec2:region:account-id:security-group/*"
            ]
        }
    ]
}
```

For more information about using tag-based policies with launch templates, see Control access to launch templates with IAM permissions in the *Amazon EC2 User Guide*.

Require a launch template and a version number

You can also use IAM permissions to enforce that a launch template and the version number of the launch template must be specified when creating or updating Auto Scaling groups.

The following example allows users to create and update Auto Scaling groups only if a launch template and the version number of the launch template are specified. If users with this policy omit the version number to specify either the `$Latest` or `$Default` launch template version, or attempt to use a launch configuration instead, the action fails.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:UpdateAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": { "autoscaling:LaunchTemplateVersionSpecified": "true" }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:UpdateAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "Null": { "autoscaling:LaunchConfigurationName": "false" }
            }
        }
    ]
}
```

Require the use of instance metadata service version 2 (IMDSv2)

For extra security, you can set your users' permissions to require the use of a launch template that requires IMDSv2. For more information, see Configuring the instance metadata service in the *Amazon EC2 User Guide*.

The following example specifies that users can't call the `ec2:RunInstances` action unless the instance is also opted in to require the use of IMDSv2 (indicated by `"ec2:MetadataHttpTokens":"required"`).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RequireImdsV2",
            "Effect": "Deny",
            "Action": "ec2:RunInstances",
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringNotEquals": { "ec2:MetadataHttpTokens": "required" }
            }
        }
    ]
}
```

> **Tip**
>
> To force replacement Auto Scaling instances to launch that use a new launch template or a new version of a launch template with the instance metadata options configured, you can start an instance refresh. For more information, see Update Auto Scaling instances.

Restrict access to Amazon EC2 resources

The following example controls the configuration of the instances that a user can launch by restricting access to Amazon EC2 resources. To specify resource-level permissions for resources specified in a launch template, you must include the resources in the `RunInstances` action statement.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:region:account-id:launch-template/*",
                "arn:aws:ec2:region::image/ami-04d5cc9b88example",
                "arn:aws:ec2:region:account-id:subnet/subnet-1a2b3c4d",
                "arn:aws:ec2:region:account-id:volume/*",
                "arn:aws:ec2:region:account-id:key-pair/*",
                "arn:aws:ec2:region:account-id:network-interface/*",
                "arn:aws:ec2:region:account-id:security-group/sg-903004f88example"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": "arn:aws:ec2:region:account-id:instance/*",
            "Condition": {
                "StringEquals": { "ec2:InstanceType": ["t2.micro", "t2.small"] }
            }
        }
    ]
}
```

In this example, there are two statements:

- The first statement requires that users launch instances into a specific subnet (**`subnet-1a2b3c4d`**), using a specific security group (**`sg-903004f88example`**), and using a specific AMI (**`ami-04d5cc9b88example`**). It also gives users access to the resources specified in a launch template: network interfaces, key pairs, and volumes.
- The second statement allows users to launch instances using only the **`t2.micro`** and **`t2.small`** instance types, which you might do to control costs.

  However, note that there is not currently an effective way to completely prevent users who have permission to launch instances with a launch template from launching other instance types. This is because an instance type specified in a launch template can be overridden to use instance types that are defined using attribute-based instance type selection.

For a full list of the resource-level permissions that you can use to control the configuration of the instances that a user can launch, see Actions, resources, and condition keys for Amazon EC2 in the *Service Authorization Reference*.

Permissions required to tag instances and volumes

The following example allows users to tag instances and volumes on creation. This policy is needed if there are tags specified in the launch template. For more information, see Grant permission to tag resources during creation in the *Amazon EC2 User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:region:account-id:*/*",
            "Condition": {
                "StringEquals": { "ec2:CreateAction": "RunInstances" }
            }
        }
    ]
}
```

Additional launch template permissions

You must give your console users permissions for the `ec2:DescribeLaunchTemplates` and `ec2:DescribeLaunchTemplateVersions` actions. Without these permissions, launch template data cannot load in the Auto Scaling group wizard, and users cannot step through the wizard to launch instances using a launch template. You can specify these additional actions in the `Action` element of an IAM policy statement.

Permissions validation for `ec2:RunInstances` and `iam:PassRole`

Users can specify which version of a launch template their Auto Scaling group uses. Depending on their permissions, this can be a specific numbered version, or the `$Latest` or `$Default` version of the launch template. If it's the latter, take special care. This may override permissions for `ec2:RunInstances` and `iam:PassRole` that you intended to restrict.

This section explains the scenario of using the latest or default version of the launch template with an Auto Scaling group.

When a user calls the `CreateAutoScalingGroup`, `UpdateAutoScalingGroup`, or `StartInstanceRefresh` APIs, Amazon EC2 Auto Scaling checks their permissions against the version of the launch template that is the latest or default version at that time before proceeding with the request. This validates permissions for actions to be completed when launching instances, such as the `ec2:RunInstances` and `iam:PassRole` actions. To accomplish this, we issue an Amazon EC2 RunInstances dry run call to validate whether the user has the required permissions for the action, without actually making the request. When a response is returned, it is read by Amazon EC2 Auto Scaling. If the user's permissions do not allow a given action, Amazon EC2 Auto Scaling fails the request and returns an error to the user containing information about the missing permission.

After the initial verification and request are complete, whenever instances launch, Amazon EC2 Auto Scaling launches them with the latest or default version, even if it has changed, using the permissions of its service-linked role. This means that a user who is using the launch template could potentially update it to pass an IAM role to an instance even if they don't have the `iam:PassRole` permission.

Use the `autoscaling:LaunchTemplateVersionSpecified` condition key if you want to limit who has access to configuring groups to use the `$Latest` or `$Default` version. This ensures that the Auto Scaling group only accepts a specific numbered version when a user calls the `CreateAutoScalingGroup` and `UpdateAutoScalingGroup` APIs. For an example that shows how to add this condition key to an IAM policy, see Require a launch template and a version number.

For Auto Scaling groups that are configured to use the `$Latest` or `$Default` launch template version, consider limiting who can create and manage versions of the launch template, including the `ec2:ModifyLaunchTemplate` action that allows a user to specify the default launch template version. For more information, see Control versioning permissions in the *Amazon EC2 User Guide*.

Related resources

To learn more about permissions to view, create, and delete launch templates and launch template versions, see Control access to launch templates with IAM permissions in the *Amazon EC2 User Guide*.

For more information about the resource-level permissions that you can use to control access to the `RunInstances` call, see Actions, resources, and condition keys for Amazon EC2 in the *Service Authorization Reference*.

IAM role for applications that run on Amazon EC2 instances

Applications that run on Amazon EC2 instances need credentials to access other AWS services. To provide these credentials in a secure way, use an IAM role. The role supplies temporary permissions that the application can use when it accesses other AWS resources. The role's permissions determine what the application is allowed to do.

For instances in an Auto Scaling group, you must create a launch template or launch configuration and choose an instance profile to associate with the instances. An instance profile is a container for an IAM role that allows Amazon EC2 to pass the IAM role to an instance when the instance is launched. First, create an IAM role that has all of the permissions required to access the AWS resources. Then, create the instance profile and assign the role to it.

> **Note**
>
> As a best practice, we strongly recommend that you create the role so that it has the minimum permissions to other AWS services that your application requires.

Contents

- Prerequisites
- Create a launch template
- See also

Prerequisites

Create the IAM role that your application running on Amazon EC2 can assume. Choose the appropriate permissions so that the application that is subsequently given the role can make the specific API calls that it needs.

If you use the IAM console instead of the AWS CLI or one of the AWS SDKs, the console creates an instance profile automatically and gives it the same name as the role to which it corresponds.

To create an IAM role (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Roles**.
3. Choose **Create role**.

4. For **Select trusted entity**, choose **AWS service**.
5. For your use case, choose **EC2** and then choose **Next**.
6. If possible, select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see Creating IAM policies in the *IAM User Guide*. After you create the policy, close that tab and return to your original tab. Select the check box next to the permissions policies that you want the service to have.
7. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
8. Choose **Next**.
9. On the **Name, review, and create** page, for **Role name**, enter a role name to help you identify the purpose of this role. This name must be unique within your AWS account. Because other AWS resources might reference the role, you can't edit the name of the role after it has been created.
10. Review the role, and then choose **Create role**.

IAM permissions

Use an IAM identity-based policy to control access to your new IAM role. The `iam:PassRole` permission is needed on the IAM identity (user or role) that creates or updates an Auto Scaling group using a launch template that specifies an instance profile.

The following example policy grants permissions to pass only IAM roles whose name begins with **`qateam-`**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account-id:role/qateam-*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com",
                        "ec2.amazonaws.com.cn"
```

```
                    ]
                }
            }
        }
    ]
}
```

> **Important**
>
> For information about how Amazon EC2 Auto Scaling validates permissions for the `iam:PassRole` action for an Auto Scaling group that uses a launch template, see Permissions validation for `ec2:RunInstances` and `iam:PassRole`.

Create a launch template

When you create the launch template using the AWS Management Console, in the **Advanced details** section, select the role from **IAM instance profile**. For more information, see Create a launch template using advanced settings.

When you create the launch template using the create-launch-template command from the AWS CLI, specify the instance profile name of your IAM role as shown in the following example.

```
aws ec2 create-launch-template --launch-template-name my-lt-with-instance-profile --
version-description version1 \
--launch-template-data
 '{"ImageId":"ami-04d5cc9b88example","InstanceType":"t2.micro","IamInstanceProfile":
{"Name":"my-instance-profile"}}'
```

See also

For more information to help you start learning about and using IAM roles for Amazon EC2, see:

- IAM roles for Amazon EC2 in the *Amazon EC2 User Guide*
- Using instance profiles and Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*

Compliance validation for Amazon EC2 Auto Scaling

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

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

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your

compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.

- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

PCI DSS compliance

Amazon EC2 Auto Scaling supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see PCI DSS Level 1.

For information on achieving PCI DSS compliance for your AWS workloads, refer to the following compliance guide:

- Payment Card Industry Data Security Standard (PCI DSS) 3.2.1 on AWS

Amazon EC2 Auto Scaling and interface VPC endpoints

You can improve the security posture of your VPC by configuring Amazon EC2 Auto Scaling to use an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Amazon EC2 Auto Scaling APIs by restricting all network traffic between your VPC and Amazon EC2 Auto Scaling to the AWS network. With interface endpoints, you also don't need an internet gateway, a NAT device, or a virtual private gateway.

You are not required to configure AWS PrivateLink, but it's recommended. For more information about AWS PrivateLink and VPC endpoints, see What is AWS PrivateLink? in the *AWS PrivateLink Guide*.

Topics

- Create an interface VPC endpoint
- Create a VPC endpoint policy

Create an interface VPC endpoint

Create an endpoint for Amazon EC2 Auto Scaling using the following service name:

```
com.amazonaws.region.autoscaling
```

For more information, see Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

You do not need to change any Amazon EC2 Auto Scaling settings. Amazon EC2 Auto Scaling calls other AWS services using either service endpoints or private interface VPC endpoints, whichever are in use.

Create a VPC endpoint policy

You can attach a policy to your VPC endpoint to control access to the Amazon EC2 Auto Scaling API. The policy specifies:

- The principal that can perform actions.
- The actions that can be performed.
- The resource on which the actions can be performed.

The following example shows a VPC endpoint policy that denies everyone permission to delete a scaling policy through the endpoint. The example policy also grants everyone permission to perform all other actions.

```
{
   "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "autoscaling:DeleteScalingPolicy",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
```

```
        }
    ]
}
```

For more information, see Control access to VPC endpoints using endpoint policies in the *AWS PrivateLink Guide*.

Using this service with an AWS SDK

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

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to this service, see Code examples for Auto Scaling using AWS SDKs.

Example availability

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

Code examples for Auto Scaling using AWS SDKs

The following code examples show how to use Auto Scaling with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello Auto Scaling

The following code examples show how to get started using Auto Scaling.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;

public class HelloAutoScaling
```

```
{
    /// <summary>
    /// Hello Amazon EC2 Auto Scaling. List EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="args"></param>
    /// <returns>Async Task.</returns>
    static async Task Main(string[] args)
    {
        var client = new AmazonAutoScalingClient();

        Console.WriteLine("Welcome to Amazon EC2 Auto Scaling.");
        Console.WriteLine("Let's get a description of your Auto Scaling
 groups.");

        var response = await client.DescribeAutoScalingGroupsAsync();

        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availabil
        });

        if (response.AutoScalingGroups.Count == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
        }
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS autoscaling)

# Set this project's name.
project("hello_autoscaling")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_autoscaling.cpp)
```

```
target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the hello_autoscaling.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/autoscaling/AutoScalingClient.h>
#include <aws/autoscaling/model/DescribeAutoScalingGroupsRequest.h>
#include <iostream>

/*
 *  A "Hello Autoscaling" starter application which initializes an Amazon EC2
 Auto Scaling client and describes the
 *  Amazon EC2 Auto Scaling groups.
 *
 *  main function
 *
 *  Usage: 'hello_autoscaling'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::AutoScaling::AutoScalingClient autoscalingClient(clientConfig);

        std::vector<Aws::String> groupNames;
        Aws::String nextToken; // Used for pagination.

        do {

            Aws::AutoScaling::Model::DescribeAutoScalingGroupsRequest request;
            if (!nextToken.empty()) {
                request.SetNextToken(nextToken);
```

```
            }

            Aws::AutoScaling::Model::DescribeAutoScalingGroupsOutcome outcome =
                    autoscalingClient.DescribeAutoScalingGroups(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::AutoScaling::Model::AutoScalingGroup>
 &autoScalingGroups =
                        outcome.GetResult().GetAutoScalingGroups();
                for (auto &group: autoScalingGroups) {
                    groupNames.push_back(group.GetAutoScalingGroupName());
                }
                nextToken = outcome.GetResult().GetNextToken();
            } else {
                std::cerr << "Error with AutoScaling::DescribeAutoScalingGroups.
 "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "Found " << groupNames.size() << " AutoScaling groups." <<
 std::endl;
        for (auto &groupName: groupNames) {
            std::cout << "AutoScaling group: " << groupName << std::endl;
        }

    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.autoscaling.AutoScalingClient;
import software.amazon.awssdk.services.autoscaling.model.AutoScalingGroup;
import
 software.amazon.awssdk.services.autoscaling.model.DescribeAutoScalingGroupsResponse;
import java.util.List;

/**
 * Before running this SDK for Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeAutoScalingGroups {
    public static void main(String[] args) throws InterruptedException {
        AutoScalingClient autoScalingClient = AutoScalingClient.builder()
                .region(Region.US_EAST_1)
                .build();

        describeGroups(autoScalingClient);
    }

    public static void describeGroups(AutoScalingClient autoScalingClient) {
        DescribeAutoScalingGroupsResponse response =
 autoScalingClient.describeAutoScalingGroups();
        List<AutoScalingGroup> groups = response.autoScalingGroups();
        groups.forEach(group -> {
            System.out.println("Group Name: " + group.autoScalingGroupName());
            System.out.println("Group ARN: " + group.autoScalingGroupARN());
```

```
        });
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Java 2.x API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function helloService()
    {
        $autoScalingClient = new AutoScalingClient([
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ]);

        $groups = $autoScalingClient->describeAutoScalingGroups([]);
        var_dump($groups);
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import boto3


def hello_autoscaling(autoscaling_client):
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon EC2 Auto Scaling
 client and list
    some of the Auto Scaling groups in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client object.
    """
    print(
        "Hello, Amazon EC2 Auto Scaling! Let's list up to ten of you Auto Scaling
 groups:"
    )
    response = autoscaling_client.describe_auto_scaling_groups()
    groups = response.get("AutoScalingGroups", [])
    if groups:
        for group in groups:
            print(f"\t{group['AutoScalingGroupName']}:
 {group['AvailabilityZones']}")
    else:
        print("There are no Auto Scaling groups in your account.")


if __name__ == "__main__":
    hello_autoscaling(boto3.client("autoscaling"))
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-autoscaling'
require 'logger'

# AutoScalingManager is a class responsible for managing AWS Auto Scaling
 operations
# such as listing all Auto Scaling groups in the current AWS account.
class AutoScalingManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Gets and prints a list of Auto Scaling groups for the account.
  def list_auto_scaling_groups
    paginator = @client.describe_auto_scaling_groups
    auto_scaling_groups = []
    paginator.each_page do |page|
      auto_scaling_groups.concat(page.auto_scaling_groups)
    end

    if auto_scaling_groups.empty?
      @logger.info('No Auto Scaling groups found for this account.')
    else
      auto_scaling_groups.each do |group|
        @logger.info("Auto Scaling group name: #{group.auto_scaling_group_name}")
        @logger.info("  Group ARN:             #{group.auto_scaling_group_arn}")
```

```
        @logger.info("  Min/max/desired:       #{group.min_size}/
#{group.max_size}/#{group.desired_capacity}")
        @logger.info("\n")
      end
    end
  end
end

if $PROGRAM_NAME == __FILE__
  autoscaling_client = Aws::AutoScaling::Client.new
  manager = AutoScalingManager.new(autoscaling_client)
  manager.list_auto_scaling_groups
end
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn list_groups(client: &Client) -> Result<(), Error> {
    let resp = client.describe_auto_scaling_groups().send().await?;

    println!("Groups:");

    let groups = resp.auto_scaling_groups();

    for group in groups {
        println!(
            "Name:  {}",
            group.auto_scaling_group_name().unwrap_or("Unknown")
        );
        println!(
            "Arn:   {}",
```

```
            group.auto_scaling_group_arn().unwrap_or("unknown"),
        );
        println!("Zones: {:?}", group.availability_zones(),);
        println!();
    }

    println!("Found {} group(s)", groups.len());

    Ok(())
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Rust API reference*.

Code examples

- Basic examples for Auto Scaling using AWS SDKs
  - Hello Auto Scaling
  - Learn the basics of Auto Scaling with an AWS SDK
  - Actions for Auto Scaling using AWS SDKs
    - Use AttachInstances with an AWS SDK or CLI
    - Use AttachLoadBalancerTargetGroups with an AWS SDK or CLI
    - Use AttachLoadBalancers with an AWS SDK or CLI
    - Use CompleteLifecycleAction with an AWS SDK or CLI
    - Use CreateAutoScalingGroup with an AWS SDK or CLI
    - Use CreateLaunchConfiguration with an AWS SDK or CLI
    - Use CreateOrUpdateTags with an AWS SDK or CLI
    - Use DeleteAutoScalingGroup with an AWS SDK or CLI
    - Use DeleteLaunchConfiguration with an AWS SDK or CLI
    - Use DeleteLifecycleHook with an AWS SDK or CLI
    - Use DeleteNotificationConfiguration with an AWS SDK or CLI
    - Use DeletePolicy with an AWS SDK or CLI
    - Use DeleteScheduledAction with an AWS SDK or CLI
    - Use DeleteTags with an AWS SDK or CLI
    - Use DescribeAccountLimits with an AWS SDK or CLI

- Use DescribeAdjustmentTypes with an AWS SDK or CLI
- Use DescribeAutoScalingGroups with an AWS SDK or CLI
- Use DescribeAutoScalingInstances with an AWS SDK or CLI
- Use DescribeAutoScalingNotificationTypes with an AWS SDK or CLI
- Use DescribeLaunchConfigurations with an AWS SDK or CLI
- Use DescribeLifecycleHookTypes with an AWS SDK or CLI
- Use DescribeLifecycleHooks with an AWS SDK or CLI
- Use DescribeLoadBalancers with an AWS SDK or CLI
- Use DescribeMetricCollectionTypes with an AWS SDK or CLI
- Use DescribeNotificationConfigurations with an AWS SDK or CLI
- Use DescribePolicies with an AWS SDK or CLI
- Use DescribeScalingActivities with an AWS SDK or CLI
- Use DescribeScalingProcessTypes with an AWS SDK or CLI
- Use DescribeScheduledActions with an AWS SDK or CLI
- Use DescribeTags with an AWS SDK or CLI
- Use DescribeTerminationPolicyTypes with an AWS SDK or CLI
- Use DetachInstances with an AWS SDK or CLI
- Use DetachLoadBalancers with an AWS SDK or CLI
- Use DisableMetricsCollection with an AWS SDK or CLI
- Use EnableMetricsCollection with an AWS SDK or CLI
- Use EnterStandby with an AWS SDK or CLI
- Use ExecutePolicy with an AWS SDK or CLI
- Use ExitStandby with an AWS SDK or CLI
- Use PutLifecycleHook with an AWS SDK or CLI
- Use PutNotificationConfiguration with an AWS SDK or CLI
- Use PutScalingPolicy with an AWS SDK or CLI
- Use PutScheduledUpdateGroupAction with an AWS SDK or CLI
- Use RecordLifecycleActionHeartbeat with an AWS SDK or CLI
- Use ResumeProcesses with an AWS SDK or CLI

- Use SetDesiredCapacity with an AWS SDK or CLI

    - Use SetInstanceHealth with an AWS SDK or CLI
    - Use SetInstanceProtection with an AWS SDK or CLI
    - Use SuspendProcesses with an AWS SDK or CLI
    - Use TerminateInstanceInAutoScalingGroup with an AWS SDK or CLI
    - Use UpdateAutoScalingGroup with an AWS SDK or CLI
- Scenarios for Auto Scaling using AWS SDKs
    - Build and manage a resilient service using an AWS SDK

Basic examples for Auto Scaling using AWS SDKs

The following code examples show how to use the basics of Amazon EC2 Auto Scaling with AWS SDKs.

Examples

- Hello Auto Scaling
- Learn the basics of Auto Scaling with an AWS SDK
- Actions for Auto Scaling using AWS SDKs
    - Use AttachInstances with an AWS SDK or CLI
    - Use AttachLoadBalancerTargetGroups with an AWS SDK or CLI
    - Use AttachLoadBalancers with an AWS SDK or CLI
    - Use CompleteLifecycleAction with an AWS SDK or CLI
    - Use CreateAutoScalingGroup with an AWS SDK or CLI
    - Use CreateLaunchConfiguration with an AWS SDK or CLI
    - Use CreateOrUpdateTags with an AWS SDK or CLI
    - Use DeleteAutoScalingGroup with an AWS SDK or CLI
    - Use DeleteLaunchConfiguration with an AWS SDK or CLI
    - Use DeleteLifecycleHook with an AWS SDK or CLI
    - Use DeleteNotificationConfiguration with an AWS SDK or CLI
    - Use DeletePolicy with an AWS SDK or CLI
    - Use DeleteScheduledAction with an AWS SDK or CLI
    - Use DeleteTags with an AWS SDK or CLI

- Use DescribeAccountLimits with an AWS SDK or CLI
- Use DescribeAdjustmentTypes with an AWS SDK or CLI
- Use DescribeAutoScalingGroups with an AWS SDK or CLI
- Use DescribeAutoScalingInstances with an AWS SDK or CLI
- Use DescribeAutoScalingNotificationTypes with an AWS SDK or CLI
- Use DescribeLaunchConfigurations with an AWS SDK or CLI
- Use DescribeLifecycleHookTypes with an AWS SDK or CLI
- Use DescribeLifecycleHooks with an AWS SDK or CLI
- Use DescribeLoadBalancers with an AWS SDK or CLI
- Use DescribeMetricCollectionTypes with an AWS SDK or CLI
- Use DescribeNotificationConfigurations with an AWS SDK or CLI
- Use DescribePolicies with an AWS SDK or CLI
- Use DescribeScalingActivities with an AWS SDK or CLI
- Use DescribeScalingProcessTypes with an AWS SDK or CLI
- Use DescribeScheduledActions with an AWS SDK or CLI
- Use DescribeTags with an AWS SDK or CLI
- Use DescribeTerminationPolicyTypes with an AWS SDK or CLI
- Use DetachInstances with an AWS SDK or CLI
- Use DetachLoadBalancers with an AWS SDK or CLI
- Use DisableMetricsCollection with an AWS SDK or CLI
- Use EnableMetricsCollection with an AWS SDK or CLI
- Use EnterStandby with an AWS SDK or CLI
- Use ExecutePolicy with an AWS SDK or CLI
- Use ExitStandby with an AWS SDK or CLI
- Use PutLifecycleHook with an AWS SDK or CLI
- Use PutNotificationConfiguration with an AWS SDK or CLI
- Use PutScalingPolicy with an AWS SDK or CLI
- Use PutScheduledUpdateGroupAction with an AWS SDK or CLI
- Use RecordLifecycleActionHeartbeat with an AWS SDK or CLI

- Use ResumeProcesses with an AWS SDK or CLI

- Use SetDesiredCapacity with an AWS SDK or CLI
- Use SetInstanceHealth with an AWS SDK or CLI
- Use SetInstanceProtection with an AWS SDK or CLI
- Use SuspendProcesses with an AWS SDK or CLI
- Use TerminateInstanceInAutoScalingGroup with an AWS SDK or CLI
- Use UpdateAutoScalingGroup with an AWS SDK or CLI

Hello Auto Scaling

The following code examples show how to get started using Auto Scaling.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;

public class HelloAutoScaling
{
    /// <summary>
    /// Hello Amazon EC2 Auto Scaling. List EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="args"></param>
    /// <returns>Async Task.</returns>
    static async Task Main(string[] args)
    {
        var client = new AmazonAutoScalingClient();

        Console.WriteLine("Welcome to Amazon EC2 Auto Scaling.");
```

```
        Console.WriteLine("Let's get a description of your Auto Scaling
 groups.");

        var response = await client.DescribeAutoScalingGroupsAsync();

        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availabil
        });

        if (response.AutoScalingGroups.Count == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
        }
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS autoscaling)

# Set this project's name.
```

```
project("hello_autoscaling")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                       # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_autoscaling.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the hello_autoscaling.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/autoscaling/AutoScalingClient.h>
```

```
#include <aws/autoscaling/model/DescribeAutoScalingGroupsRequest.h>
#include <iostream>

/*
 *  A "Hello Autoscaling" starter application which initializes an Amazon EC2
 Auto Scaling client and describes the
 *  Amazon EC2 Auto Scaling groups.
 *
 *  main function
 *
 *  Usage: 'hello_autoscaling'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::AutoScaling::AutoScalingClient autoscalingClient(clientConfig);

        std::vector<Aws::String> groupNames;
        Aws::String nextToken; // Used for pagination.

        do {

            Aws::AutoScaling::Model::DescribeAutoScalingGroupsRequest request;
            if (!nextToken.empty()) {
                request.SetNextToken(nextToken);
            }

            Aws::AutoScaling::Model::DescribeAutoScalingGroupsOutcome outcome =
                    autoscalingClient.DescribeAutoScalingGroups(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::AutoScaling::Model::AutoScalingGroup>
 &autoScalingGroups =
                        outcome.GetResult().GetAutoScalingGroups();
```

```
                for (auto &group: autoScalingGroups) {
                    groupNames.push_back(group.GetAutoScalingGroupName());
                }
                nextToken = outcome.GetResult().GetNextToken();
            } else {
                std::cerr << "Error with AutoScaling::DescribeAutoScalingGroups.
 "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "Found " << groupNames.size() << " AutoScaling groups." <<
 std::endl;
        for (auto &groupName: groupNames) {
            std::cout << "AutoScaling group: " << groupName << std::endl;
        }

    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.autoscaling.AutoScalingClient;
```

```
import software.amazon.awssdk.services.autoscaling.model.AutoScalingGroup;
import
 software.amazon.awssdk.services.autoscaling.model.DescribeAutoScalingGroupsResponse;
import java.util.List;

/**
 * Before running this SDK for Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeAutoScalingGroups {
    public static void main(String[] args) throws InterruptedException {
        AutoScalingClient autoScalingClient = AutoScalingClient.builder()
                .region(Region.US_EAST_1)
                .build();

        describeGroups(autoScalingClient);
    }

    public static void describeGroups(AutoScalingClient autoScalingClient) {
        DescribeAutoScalingGroupsResponse response =
 autoScalingClient.describeAutoScalingGroups();
        List<AutoScalingGroup> groups = response.autoScalingGroups();
        groups.forEach(group -> {
            System.out.println("Group Name: " + group.autoScalingGroupName());
            System.out.println("Group ARN: " + group.autoScalingGroupARN());
        });
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Java 2.x API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function helloService()
    {
        $autoScalingClient = new AutoScalingClient([
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ]);

        $groups = $autoScalingClient->describeAutoScalingGroups([]);
        var_dump($groups);
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import boto3


def hello_autoscaling(autoscaling_client):
    """
```

```
    Use the AWS SDK for Python (Boto3) to create an Amazon EC2 Auto Scaling
 client and list
    some of the Auto Scaling groups in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client object.
    """
    print(
        "Hello, Amazon EC2 Auto Scaling! Let's list up to ten of you Auto Scaling
 groups:"
    )
    response = autoscaling_client.describe_auto_scaling_groups()
    groups = response.get("AutoScalingGroups", [])
    if groups:
        for group in groups:
            print(f"\t{group['AutoScalingGroupName']}:
 {group['AvailabilityZones']}")
    else:
        print("There are no Auto Scaling groups in your account.")


if __name__ == "__main__":
    hello_autoscaling(boto3.client("autoscaling"))
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-autoscaling'
```

```
require 'logger'

# AutoScalingManager is a class responsible for managing AWS Auto Scaling
 operations
# such as listing all Auto Scaling groups in the current AWS account.
class AutoScalingManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Gets and prints a list of Auto Scaling groups for the account.
  def list_auto_scaling_groups
    paginator = @client.describe_auto_scaling_groups
    auto_scaling_groups = []
    paginator.each_page do |page|
      auto_scaling_groups.concat(page.auto_scaling_groups)
    end

    if auto_scaling_groups.empty?
      @logger.info('No Auto Scaling groups found for this account.')
    else
      auto_scaling_groups.each do |group|
        @logger.info("Auto Scaling group name: #{group.auto_scaling_group_name}")
        @logger.info("  Group ARN:             #{group.auto_scaling_group_arn}")
        @logger.info("  Min/max/desired:       #{group.min_size}/
#{group.max_size}/#{group.desired_capacity}")
        @logger.info("\n")
      end
    end
  end
end

if $PROGRAM_NAME == __FILE__
  autoscaling_client = Aws::AutoScaling::Client.new
  manager = AutoScalingManager.new(autoscaling_client)
  manager.list_auto_scaling_groups
end
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn list_groups(client: &Client) -> Result<(), Error> {
    let resp = client.describe_auto_scaling_groups().send().await?;

    println!("Groups:");

    let groups = resp.auto_scaling_groups();

    for group in groups {
        println!(
            "Name:  {}",
            group.auto_scaling_group_name().unwrap_or("Unknown")
        );
        println!(
            "Arn:   {}",
            group.auto_scaling_group_arn().unwrap_or("unknown"),
        );
        println!("Zones: {:?}", group.availability_zones(),);
        println!();
    }

    println!("Found {} group(s)", groups.len());

    Ok(())
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Learn the basics of Auto Scaling with an AWS SDK

The following code examples show how to:

- Create an Amazon EC2 Auto Scaling group with a launch template and Availability Zones, and get information about running instances.
- Enable Amazon CloudWatch metrics collection.
- Update the group's desired capacity and wait for an instance to start.
- Terminate an instance in the group.
- List scaling activities that occur in response to user requests and capacity changes.
- Get statistics for CloudWatch metrics, then clean up resources.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
global using Amazon.AutoScaling;
global using Amazon.AutoScaling.Model;
global using Amazon.CloudWatch;
global using AutoScalingActions;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.EC2;
using Microsoft.Extensions.Configuration;
using Host = Microsoft.Extensions.Hosting.Host;

namespace AutoScalingBasics;
```

```
public class AutoScalingBasics
{

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 Auto Scaling, Amazon
        // CloudWatch, and Amazon EC2.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonCloudWatch>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalingWrapper>()
                .AddTransient<CloudWatchWrapper>()
                .AddTransient<EC2Wrapper>()
                .AddTransient<UIWrapper>()
            )
            .Build();


        var autoScalingWrapper =
 host.Services.GetRequiredService<AutoScalingWrapper>();
        var cloudWatchWrapper =
 host.Services.GetRequiredService<CloudWatchWrapper>();
        var ec2Wrapper = host.Services.GetRequiredService<EC2Wrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var imageId = configuration["ImageId"];
        var instanceType = configuration["InstanceType"];
        var launchTemplateName = configuration["LaunchTemplateName"];
```

```
        launchTemplateName += Guid.NewGuid().ToString();

        // The name of the Auto Scaling group.
        var groupName = configuration["GroupName"];

        uiWrapper.DisplayTitle("Auto Scaling Basics");
        uiWrapper.DisplayAutoScalingBasicsDescription();

        // Create the launch template and save the template Id to use when
deleting the
        // launch template at the end of the application.
        var launchTemplateId = await
ec2Wrapper.CreateLaunchTemplateAsync(imageId!, instanceType!,
launchTemplateName);

        // Confirm that the template was created by asking for a description of
it.
        await ec2Wrapper.DescribeLaunchTemplateAsync(launchTemplateName);

        uiWrapper.PressEnter();

        var availabilityZones = await ec2Wrapper.ListAvailabilityZonesAsync();

        Console.WriteLine($"Creating an Auto Scaling group named {groupName}.");
        await autoScalingWrapper.CreateAutoScalingGroupAsync(
            groupName!,
            launchTemplateName,
            availabilityZones.First().ZoneName);

        // Keep checking the details of the new group until its lifecycle state
        // is "InService".
        Console.WriteLine($"Waiting for the Auto Scaling group to be active.");

        List<AutoScalingInstanceDetails> instanceDetails;

        do
        {
            instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
        }
        while (instanceDetails.Count <= 0);
```

```
        Console.WriteLine($"Auto scaling group {groupName} successfully
created.");
        Console.WriteLine($"{instanceDetails.Count} instances were created for
the group.");

        // Display the details of the Auto Scaling group.
        instanceDetails.ForEach(detail =>
        {
            Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Metrics collection");
        Console.WriteLine($"Enable metrics collection for {groupName}");
        await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);

        // Show the metrics that are collected for the group.

        // Update the maximum size of the group to three instances.
        Console.WriteLine("--- Update the Auto Scaling group to increase max size
to 3 ---");
        int maxSize = 3;
        await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

        Console.WriteLine("--- Describe all Auto Scaling groups to show the
current state of the group ---");
        var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

        uiWrapper.DisplayGroupDetails(groups!);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");

        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);
```

```
        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups!.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups is not null)
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is
{group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
                {
                    Console.WriteLine($"The instance id is
{instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred
for the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities is not null)
        {
            activities.ForEach(activity =>
            {
                Console.WriteLine($"The activity Id is {activity.ActivityId}");
                Console.WriteLine($"The activity details are
{activity.Details}");
            });
        }

        // Display the Amazon CloudWatch metrics that have been collected.
        var metrics = await
cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
```

```
        Console.WriteLine($"Metrics collected for {groupName}:");
        metrics.ForEach(metric =>
        {
            Console.Write($"Metric name: {metric.MetricName}\t");
            Console.WriteLine($"Namespace: {metric.Namespace}");
        });

        var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
        Console.WriteLine("Details for the metrics collected:");
        dataPoints.ForEach(detail =>
        {
            Console.WriteLine(detail);
        });

        // Disable metrics collection.
        Console.WriteLine("Disabling the collection of metrics for
{groupName}.");
        var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

        if (success)
        {
            Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
        }
        else
        {
            Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
        }

        // Terminate all instances in the group.
        uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
        Console.WriteLine("Now terminating all instances in the Auto Scaling
group.");

        if (groups is not null)
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
                if (group.AutoScalingGroupName == groupName)
                {
```

```
                    group.Instances.ForEach(async instance =>
                    {
                        await
 autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
 ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);

        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}


namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
    {
```

```
        Console.WriteLine("This code example performs the following
operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto
Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the
group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred
for the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that
have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;

        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
```

```
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
```

```
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Define functions that are called by the scenario to manage launch templates and metrics. These functions wrap Auto Scaling, Amazon EC2, and CloudWatch actions.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;

    /// <summary>
    /// Constructor for the AutoScalingWrapper class.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
 client.</param>
    public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
    {
        _amazonAutoScaling = amazonAutoScaling;
    }
```

```
    /// <summary>
    /// Create a new Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name to use for the new Auto Scaling
    /// group.</param>
    /// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
    /// launch template to use to create instances in the group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        string availabilityZone)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var zoneList = new List<string>
            {
                availabilityZone,
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };

        var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
```

```
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
 response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
 response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Amazon.AutoScaling.Model.Activity>>
 DescribeScalingActivitiesAsync(
        string groupName)
    {
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        return response.Activities;
    }



    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
```

```
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }



    /// <summary>
    /// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
    public async Task<List<AutoScalingGroup>?> DescribeAutoScalingGroupsAsync(
        string groupName)
    {
        var groupList = new List<string>
            {
                groupName,
```

```
            };

        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        var groups = response.AutoScalingGroups;

        return groups;
    }


    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }


    /// <summary>
```

```
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
 _amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
 {desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };
```

```
        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }


    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
```

```
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }

}


namespace AutoScalingActions;

using Amazon.EC2;
using Amazon.EC2.Model;

public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
    {
```

```
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }

    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string
 launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
```

```
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }

        return false;
    }

    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}


namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
```

```
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A list of Metrics collected for the Auto Scaling group.</
returns>
    public async Task<List<Amazon.CloudWatch.Model.Metric>>
 GetCloudWatchMetricsAsync(string groupName)
    {
        var filter = new DimensionFilter
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        };

        var request = new ListMetricsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = new List<DimensionFilter> { filter },
            Namespace = "AWS/AutoScaling",
        };

        var response = await _amazonCloudWatch.ListMetricsAsync(request);

        return response.Metrics;
    }

    /// <summary>
    /// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of data points.</returns>
```

```
    public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
    {
        var metricDimensions = new List<Dimension>
            {
                new Dimension
                {
                    Name = "AutoScalingGroupName",
                    Value = $"{groupName}",
                },
            };

        // The start time will be yesterday.
        var startTime = DateTime.UtcNow.AddDays(-1);

        var request = new GetMetricStatisticsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = metricDimensions,
            Namespace = "AWS/AutoScaling",
            Period = 60, // 60 seconds.
            Statistics = new List<string>() { "Minimum" },
            StartTimeUtc = startTime,
            EndTimeUtc = DateTime.UtcNow,
        };

        var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);

        return response.Datapoints;
    }

}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection

- EnableMetricsCollection
- SetDesiredCapacity
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
//! Routine which demonstrates using an Auto Scaling group
//! to manage Amazon EC2 instances.
/*!
  \sa groupsAndInstancesScenario()
  \param clientConfig: AWS client configuration.
  \return bool: Successful completion.
 */
bool AwsDoc::AutoScaling::groupsAndInstancesScenario(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::String templateName;
    Aws::EC2::EC2Client ec2Client(clientConfig);

    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout
            << "Welcome to the Amazon Elastic Compute Cloud (Amazon EC2) Auto
 Scaling "
            << "demo for managing groups and instances." << std::endl;
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " \n"
              << std::endl;

    std::cout << "This example requires an EC2 launch template." << std::endl;
    if (askYesNoQuestion(
            "Would you like to use an existing EC2 launch template (y/n)?  ")) {
```

```
        // 1. Specify the name of an existing EC2 launch template.
        templateName = askQuestion(
                "Enter the name of the existing EC2 launch template.  ");

        Aws::EC2::Model::DescribeLaunchTemplatesRequest request;
        request.AddLaunchTemplateNames(templateName);
        Aws::EC2::Model::DescribeLaunchTemplatesOutcome outcome =
                ec2Client.DescribeLaunchTemplates(request);

        if (outcome.IsSuccess()) {
            std::cout << "Validated the EC2 launch template '" << templateName
                      << "' exists by calling DescribeLaunchTemplate." <<
std::endl;
        }
        else {
            std::cerr << "Error validating the existence of the launch template.
"
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }
    else { // 2.  Or create a new EC2 launch template.
        templateName = askQuestion("Enter the name for a new EC2 launch template:
");

        Aws::EC2::Model::CreateLaunchTemplateRequest request;
        request.SetLaunchTemplateName(templateName);

        Aws::EC2::Model::RequestLaunchTemplateData requestLaunchTemplateData;

requestLaunchTemplateData.SetInstanceType(EC2_LAUNCH_TEMPLATE_INSTANCE_TYPE);
        requestLaunchTemplateData.SetImageId(EC2_LAUNCH_TEMPLATE_IMAGE_ID);

        request.SetLaunchTemplateData(requestLaunchTemplateData);

        Aws::EC2::Model::CreateLaunchTemplateOutcome outcome =
                ec2Client.CreateLaunchTemplate(request);

        if (outcome.IsSuccess()) {
            std::cout << "The EC2 launch template '" << templateName << " was
created."
                      << std::endl;
        }
        else if (outcome.GetError().GetExceptionName() ==
```

```
                "InvalidLaunchTemplateName.AlreadyExistsException") {
            std::cout << "The EC2 template '" << templateName << "' already
exists"
                      << std::endl;
        }
        else {
            std::cerr << "Error with EC2::CreateLaunchTemplate. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }
    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);
    std::cout << "Let's create an Auto Scaling group." << std::endl;
    Aws::String groupName = askQuestion(
            "Enter a name for the Auto Scaling group:  ");
    // 3. Retrieve a list of EC2 Availability Zones.
    Aws::Vector<Aws::EC2::Model::AvailabilityZone> availabilityZones;
    {
        Aws::EC2::Model::DescribeAvailabilityZonesRequest request;

        Aws::EC2::Model::DescribeAvailabilityZonesOutcome outcome =
                ec2Client.DescribeAvailabilityZones(request);

        if (outcome.IsSuccess()) {
            std::cout
                    << "EC2 instances can be created in the following
Availability Zones:"
                    << std::endl;

            availabilityZones = outcome.GetResult().GetAvailabilityZones();
            for (size_t i = 0; i < availabilityZones.size(); ++i) {
                std::cout << "   " << i + 1 << ".  "
                          << availabilityZones[i].GetZoneName() << std::endl;
            }
        }
        else {
            std::cerr << "Error with EC2::DescribeAvailabilityZones. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanupResources("", templateName, autoScalingClient, ec2Client);
            return false;
        }
    }
```

```
    int availabilityZoneChoice = askQuestionForIntRange(
            "Choose an Availability Zone:  ", 1,
            static_cast<int>(availabilityZones.size()));
    // 4. Create an Auto Scaling group with the specified Availability Zone.
    {
        Aws::AutoScaling::Model::CreateAutoScalingGroupRequest request;
        request.SetAutoScalingGroupName(groupName);
        Aws::Vector<Aws::String> availabilityGroupZones;
        availabilityGroupZones.push_back(
                availabilityZones[availabilityZoneChoice - 1].GetZoneName());
        request.SetAvailabilityZones(availabilityGroupZones);
        request.SetMaxSize(1);
        request.SetMinSize(1);

        Aws::AutoScaling::Model::LaunchTemplateSpecification
launchTemplateSpecification;
        launchTemplateSpecification.SetLaunchTemplateName(templateName);
        request.SetLaunchTemplate(launchTemplateSpecification);

        Aws::AutoScaling::Model::CreateAutoScalingGroupOutcome outcome =
                autoScalingClient.CreateAutoScalingGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "Created Auto Scaling group '" << groupName << "'..."
                      << std::endl;
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::AutoScaling::AutoScalingErrors::ALREADY_EXISTS_FAULT) {
            std::cout << "Auto Scaling group '" << groupName << "' already
exists."
                      << std::endl;
        }
        else {
            std::cerr << "Error with AutoScaling::CreateAutoScalingGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanupResources("", templateName, autoScalingClient, ec2Client);
            return false;
        }
    }

    Aws::Vector<Aws::AutoScaling::Model::AutoScalingGroup> autoScalingGroups;
    if (AwsDoc::AutoScaling::describeGroup(groupName, autoScalingGroups,
                                           autoScalingClient)) {
```

```
        std::cout << "Here is the Auto Scaling group description." << std::endl;
        if (!autoScalingGroups.empty()) {
            logAutoScalingGroupInfo(autoScalingGroups);
        }
    }
    else {
        cleanupResources(groupName, templateName, autoScalingClient, ec2Client);
        return false;
    }

    std::cout
            << "Waiting for the EC2 instance in the Auto Scaling group to become
active..."
            << std::endl;
    if (!waitForInstances(groupName, autoScalingGroups, autoScalingClient)) {
        cleanupResources(groupName, templateName, autoScalingClient, ec2Client);
        return false;
    }

    bool enableMetrics = askYesNoQuestion(
            "Do you want to collect metrics about the A"
            "Auto Scaling group during this demo (y/n)?  ");
    // 7. Optionally enable metrics collection for the Auto Scaling group.
    if (enableMetrics) {
        Aws::AutoScaling::Model::EnableMetricsCollectionRequest request;
        request.SetAutoScalingGroupName(groupName);

        request.AddMetrics("GroupMinSize");
        request.AddMetrics("GroupMaxSize");
        request.AddMetrics("GroupDesiredCapacity");
        request.AddMetrics("GroupInServiceInstances");
        request.AddMetrics("GroupTotalInstances");
        request.SetGranularity("1Minute");

        Aws::AutoScaling::Model::EnableMetricsCollectionOutcome outcome =
                autoScalingClient.EnableMetricsCollection(request);
        if (outcome.IsSuccess()) {
            std::cout << "Auto Scaling metrics have been enabled."
                      << std::endl;
        }
        else {
            std::cerr << "Error with AutoScaling::EnableMetricsCollection. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    std::cout << "Let's update the maximum number of EC2 instances in '" <<
groupName <<
              "' from 1 to 3." << std::endl;
    askQuestion("Press enter to continue:  ", alwaysTrueTest);
    // 8. Update the Auto Scaling group, setting a new maximum size.
    {
        Aws::AutoScaling::Model::UpdateAutoScalingGroupRequest request;
        request.SetAutoScalingGroupName(groupName);
        request.SetMaxSize(3);

        Aws::AutoScaling::Model::UpdateAutoScalingGroupOutcome outcome =
                autoScalingClient.UpdateAutoScalingGroup(request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Error with AutoScaling::UpdateAutoScalingGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    if (AwsDoc::AutoScaling::describeGroup(groupName, autoScalingGroups,
                                           autoScalingClient)) {
        if (!autoScalingGroups.empty()) {
            const auto &instances = autoScalingGroups[0].GetInstances();
            std::cout
                    << "The group still has one running EC2 instance, but it can
have up to 3.\n"
                    << std::endl;
            logAutoScalingGroupInfo(autoScalingGroups);
        }
        else {
            std::cerr
                    << "No EC2 launch groups were retrieved from DescribeGroup
request."
                    << std::endl;
```

```
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    std::cout << "\n" << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) <<
"\n"
              << std::endl;
    std::cout << "Let's update the desired capacity in '" << groupName <<
              "' from 1 to 2." << std::endl;
    askQuestion("Press enter to continue:  ", alwaysTrueTest);
    //  9. Update the Auto Scaling group, setting a new desired capacity.
    {
        Aws::AutoScaling::Model::SetDesiredCapacityRequest request;
        request.SetAutoScalingGroupName(groupName);
        request.SetDesiredCapacity(2);

        Aws::AutoScaling::Model::SetDesiredCapacityOutcome outcome =
                autoScalingClient.SetDesiredCapacity(request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Error with AutoScaling::SetDesiredCapacityRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    if (AwsDoc::AutoScaling::describeGroup(groupName, autoScalingGroups,
                                           autoScalingClient)) {
        if (!autoScalingGroups.empty()) {
            std::cout
                    << "Here is the current state of the group." << std::endl;
            logAutoScalingGroupInfo(autoScalingGroups);
        }
        else {
            std::cerr
                    << "No EC2 launch groups were retrieved from DescribeGroup
request."
                    << std::endl;
```

```
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    std::cout << "Waiting for the new EC2 instance to start..." << std::endl;
    waitForInstances(groupName, autoScalingGroups, autoScalingClient);

    std::cout << "\n" << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) <<
"\n"
              << std::endl;

    std::cout << "Let's terminate one of the EC2 instances in " << groupName <<
"."
              << std::endl;
    std::cout << "Because the desired capacity is 2, another EC2 instance will
start "
              << "to replace the terminated EC2 instance."
              << std::endl;
    std::cout << "The currently running EC2 instances are:" << std::endl;

    if (autoScalingGroups.empty()) {
        std::cerr << "Error describing groups. No groups returned." << std::endl;
        cleanupResources(groupName, templateName, autoScalingClient, ec2Client);
        return false;
    }

    int instanceNumber = 1;
    Aws::Vector<Aws::String> instanceIDs = instancesToInstanceIDs(
            autoScalingGroups[0].GetInstances());
    for (const Aws::String &instanceID: instanceIDs) {
        std::cout << "   " << instanceNumber << ". " << instanceID << std::endl;
        ++instanceNumber;
    }

    instanceNumber = askQuestionForIntRange("Which EC2 instance do you want to
stop? ",
                                            1,

static_cast<int>(instanceIDs.size()));

    // 10. Terminate an EC2 instance in the Auto Scaling group.
    {
```

```
        Aws::AutoScaling::Model::TerminateInstanceInAutoScalingGroupRequest
request;
        request.SetInstanceId(instanceIDs[instanceNumber - 1]);
        request.SetShouldDecrementDesiredCapacity(false);

        Aws::AutoScaling::Model::TerminateInstanceInAutoScalingGroupOutcome
outcome =
                autoScalingClient.TerminateInstanceInAutoScalingGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "Waiting for EC2 instance with ID '"
                      << instanceIDs[instanceNumber - 1] << "' to terminate..."
                      << std::endl;
        }
        else {
            std::cerr << "Error with
AutoScaling::TerminateInstanceInAutoScalingGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    waitForInstances(groupName, autoScalingGroups, autoScalingClient);

    std::cout << "\n" << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) <<
"\n"
              << std::endl;
    std::cout << "Let's get a report of scaling activities for EC2 launch group
'"
              << groupName << "'."
              << std::endl;
    askQuestion("Press enter to continue:  ", alwaysTrueTest);
    // 11. Get a description of activities for the Auto Scaling group.
    {
        Aws::AutoScaling::Model::DescribeScalingActivitiesRequest request;
        request.SetAutoScalingGroupName(groupName);

        Aws::Vector<Aws::AutoScaling::Model::Activity> allActivities;
        Aws::String nextToken; // Used for pagination;
        do {
            if (!nextToken.empty()) {
```

```
                request.SetNextToken(nextToken);
            }
            Aws::AutoScaling::Model::DescribeScalingActivitiesOutcome outcome =
                    autoScalingClient.DescribeScalingActivities(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::AutoScaling::Model::Activity> &activities
=
                        outcome.GetResult().GetActivities();
                allActivities.insert(allActivities.end(), activities.begin(),
activities.end());
                nextToken  = outcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error with AutoScaling::DescribeScalingActivities.
"
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
                return false;
            }
        } while (!nextToken.empty());

        std::cout << "Found " << allActivities.size() << " activities."
                  << std::endl;
        std::cout << "Activities are ordered with the most recent first."
                  << std::endl;
        for (const Aws::AutoScaling::Model::Activity &activity: allActivities) {
            std::cout << activity.GetDescription() << std::endl;
            std::cout << activity.GetDetails() << std::endl;
        }
    }

    if (enableMetrics) {
        if (!logAutoScalingMetrics(groupName, clientConfig)) {
            cleanupResources(groupName, templateName, autoScalingClient,
ec2Client);
            return false;
        }
    }

    std::cout << "Let's  clean up." << std::endl;
    askQuestion("Press enter to continue:  ", alwaysTrueTest);
```

```
    // 13. Disable metrics collection if enabled.
    if (enableMetrics) {
        Aws::AutoScaling::Model::DisableMetricsCollectionRequest request;
        request.SetAutoScalingGroupName(groupName);

        Aws::AutoScaling::Model::DisableMetricsCollectionOutcome outcome =
                autoScalingClient.DisableMetricsCollection(request);

        if (outcome.IsSuccess()) {
            std::cout << "Metrics collection has been disabled." << std::endl;
        }
        else {
            std::cerr << "Error with AutoScaling::DisableMetricsCollection. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanupResources(groupName, templateName, autoScalingClient,
 ec2Client);
            return false;
        }
    }

    return cleanupResources(groupName, templateName, autoScalingClient,
 ec2Client);
}

//! Routine which waits for EC2 instances in an Auto Scaling group to
//! complete startup or shutdown.
/*!
 \sa waitForInstances()
 \param groupName: An Auto Scaling group name.
 \param autoScalingGroups: Vector to receive 'AutoScalingGroup' records.
 \param client: 'AutoScalingClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::AutoScaling::waitForInstances(const Aws::String &groupName,

 Aws::Vector<Aws::AutoScaling::Model::AutoScalingGroup> &autoScalingGroups,
                                            const
 Aws::AutoScaling::AutoScalingClient &client) {
    bool ready = false;
    const std::vector<Aws::String> READY_STATES = {"InService", "Terminated"};

    int count = 0;
```

```
    int desiredCapacity = 0;
    std::this_thread::sleep_for(std::chrono::seconds(4));
    while (!ready) {
        if (WAIT_FOR_INSTANCES_TIMEOUT < count) {
            std::cerr << "Wait for instance timed out." << std::endl;
            return false;
        }

        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++count;
        if (!describeGroup(groupName, autoScalingGroups, client)) {
            return false;
        }
        Aws::Vector<Aws::String> instanceIDs;
        if (!autoScalingGroups.empty()) {
            instanceIDs =
instancesToInstanceIDs(autoScalingGroups[0].GetInstances());
            desiredCapacity = autoScalingGroups[0].GetDesiredCapacity();
        }

        if (instanceIDs.empty()) {
            if (desiredCapacity == 0) {
                break;
            }
            else {
                if ((count % 5) == 0) {
                    std::cout << "No instance IDs returned for group." <<
std::endl;
                }

                continue;
            }
        }

        // 6.  Check lifecycle state of the instances using
DescribeAutoScalingInstances.
        Aws::AutoScaling::Model::DescribeAutoScalingInstancesRequest request;
        request.SetInstanceIds(instanceIDs);

        Aws::AutoScaling::Model::DescribeAutoScalingInstancesOutcome outcome =
                client.DescribeAutoScalingInstances(request);

        if (outcome.IsSuccess()) {
```

```
            const
 Aws::Vector<Aws::AutoScaling::Model::AutoScalingInstanceDetails>
 &instancesDetails =
                    outcome.GetResult().GetAutoScalingInstances();
            ready = instancesDetails.size() >= desiredCapacity;
            for (const Aws::AutoScaling::Model::AutoScalingInstanceDetails
 &details: instancesDetails) {
                if (!stringInVector(details.GetLifecycleState(), READY_STATES)) {
                    ready = false;
                    break;
                }
            }
            // Log the status while waiting.
            if (((count % 5) == 1) || ready) {
                logInstancesLifecycleState(instancesDetails);
            }
        }
        else {
            std::cerr << "Error with AutoScaling::DescribeAutoScalingInstances. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!describeGroup(groupName, autoScalingGroups, client)) {
        return false;
    }

    return true;
}

//! Routine to cleanup resources created in 'groupsAndInstancesScenario'.
/*!
 \sa cleanupResources()
 \param groupName: Optional Auto Scaling group name.
 \param templateName: Optional EC2 launch template name.
 \param autoScalingClient: 'AutoScalingClient' instance.
 \param ec2Client: 'EC2Client' instance.
\return bool: Successful completion.
 */
bool AwsDoc::AutoScaling::cleanupResources(const Aws::String &groupName,
                                           const Aws::String &templateName,
```

```
                                            const
Aws::AutoScaling::AutoScalingClient &autoScalingClient,
                                            const Aws::EC2::EC2Client &ec2Client)
{
   bool result = true;

   // 14. Delete the Auto Scaling group.
   if (!groupName.empty() &&
       (askYesNoQuestion(
               Aws::String("Delete the Auto Scaling group '") + groupName +
               "'  (y/n)?"))) {
       {
           Aws::AutoScaling::Model::UpdateAutoScalingGroupRequest request;
           request.SetAutoScalingGroupName(groupName);
           request.SetMinSize(0);
           request.SetDesiredCapacity(0);

           Aws::AutoScaling::Model::UpdateAutoScalingGroupOutcome outcome =
                   autoScalingClient.UpdateAutoScalingGroup(request);

           if (outcome.IsSuccess()) {
               std::cout
                       << "The minimum size and desired capacity of the Auto
Scaling group "
                       << "was set to zero before terminating the instances."
                       << std::endl;
           }
           else {
               std::cerr << "Error with AutoScaling::UpdateAutoScalingGroup. "
                         << outcome.GetError().GetMessage() << std::endl;
               result = false;
           }
       }

       Aws::Vector<Aws::AutoScaling::Model::AutoScalingGroup> autoScalingGroups;
       if (AwsDoc::AutoScaling::describeGroup(groupName, autoScalingGroups,
                                              autoScalingClient)) {
           if (!autoScalingGroups.empty()) {
               Aws::Vector<Aws::String> instanceIDs = instancesToInstanceIDs(
                       autoScalingGroups[0].GetInstances());
               for (const Aws::String &instanceID: instanceIDs) {

Aws::AutoScaling::Model::TerminateInstanceInAutoScalingGroupRequest request;
                   request.SetInstanceId(instanceID);
```

```
                    request.SetShouldDecrementDesiredCapacity(true);


Aws::AutoScaling::Model::TerminateInstanceInAutoScalingGroupOutcome outcome =

autoScalingClient.TerminateInstanceInAutoScalingGroup(
                                  request);

                    if (outcome.IsSuccess()) {
                        std::cout << "Initiating termination of EC2 instance '"
                                  << instanceID << "'." << std::endl;
                    }
                    else {
                        std::cerr
                                << "Error with
AutoScaling::TerminateInstanceInAutoScalingGroup. "
                                << outcome.GetError().GetMessage() << std::endl;
                        result = false;
                    }
                }
            }

            std::cout
                    << "Waiting for the EC2 instances to terminate before
deleting the "
                    << "Auto Scaling group..." << std::endl;
            waitForInstances(groupName, autoScalingGroups, autoScalingClient);
        }

        {
            Aws::AutoScaling::Model::DeleteAutoScalingGroupRequest request;
            request.SetAutoScalingGroupName(groupName);

            Aws::AutoScaling::Model::DeleteAutoScalingGroupOutcome outcome =
                    autoScalingClient.DeleteAutoScalingGroup(request);

            if (outcome.IsSuccess()) {
                std::cout << "Auto Scaling group '" << groupName << "' was
deleted."
                          << std::endl;
            }
            else {
                std::cerr << "Error with AutoScaling::DeleteAutoScalingGroup. "
                          << outcome.GetError().GetMessage()
```

```
                            << std::endl;
                result = false;
            }
        }
    }

    // 15. Delete the EC2 launch template.
    if (!templateName.empty() && (askYesNoQuestion(
            Aws::String("Delete the EC2 launch template '") + templateName +
            "' (y/n)?"))) {
        Aws::EC2::Model::DeleteLaunchTemplateRequest request;
        request.SetLaunchTemplateName(templateName);

        Aws::EC2::Model::DeleteLaunchTemplateOutcome outcome =
                ec2Client.DeleteLaunchTemplate(request);

        if (outcome.IsSuccess()) {
            std::cout << "EC2 launch template '" << templateName << "' was
 deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with EC2::DeleteLaunchTemplate. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}

//! Routine which retrieves Auto Scaling group descriptions.
/*!
 \sa describeGroup()
 \param groupName: An Auto Scaling group name.
 \param autoScalingGroups: Vector to receive 'AutoScalingGroup' records.
 \param client: 'AutoScalingClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::AutoScaling::describeGroup(const Aws::String &groupName,

 Aws::Vector<Aws::AutoScaling::Model::AutoScalingGroup> &autoScalingGroup,
```

```
                                        const Aws::AutoScaling::AutoScalingClient
 &client) {
    // 5. Retrieve a description of the Auto Scaling group.
    Aws::AutoScaling::Model::DescribeAutoScalingGroupsRequest request;
    Aws::Vector<Aws::String> groupNames;
    groupNames.push_back(groupName);
    request.SetAutoScalingGroupNames(groupNames);

    Aws::AutoScaling::Model::DescribeAutoScalingGroupsOutcome outcome =
            client.DescribeAutoScalingGroups(request);

    if (outcome.IsSuccess()) {
        autoScalingGroup = outcome.GetResult().GetAutoScalingGroups();
    }
    else {
        std::cerr << "Error with AutoScaling::DescribeAutoScalingGroups. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
 * Before running this SDK for Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * In addition, create a launch template. For more information, see the
 * following topic:
 *
 * https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-launch-
templates.html#create-launch-template
 *
 * This code example performs the following operations:
 * 1. Creates an Auto Scaling group using an AutoScalingWaiter.
 * 2. Gets a specific Auto Scaling group and returns an instance Id value.
 * 3. Describes Auto Scaling with the Id value.
 * 4. Enables metrics collection.
 * 5. Update an Auto Scaling group.
 * 6. Describes Account details.
 * 7. Describe account details"
 * 8. Updates an Auto Scaling group to use an additional instance.
 * 9. Gets the specific Auto Scaling group and gets the number of instances.
 * 10. List the scaling activities that have occurred for the group.
 * 11. Terminates an instance in the Auto Scaling group.
 * 12. Stops the metrics collection.
 * 13. Deletes the Auto Scaling group.
 */

public class AutoScalingScenario {
```

```
   public static final String DASHES = new String(new char[80]).replace("\0",
"-");

   public static void main(String[] args) throws InterruptedException {
       final String usage = """

               Usage:
                   <groupName> <launchTemplateName> <vpcZoneId>

               Where:
                   groupName - The name of the Auto Scaling group.
                   launchTemplateName - The name of the launch template.\s
                   vpcZoneId - A subnet Id for a virtual private cloud (VPC)
where instances in the Auto Scaling group can be created.
               """;

       if (args.length != 3) {
           System.out.println(usage);
           System.exit(1);
       }

       String groupName = args[0];
       String launchTemplateName = args[1];
       String vpcZoneId = args[2];
       AutoScalingClient autoScalingClient = AutoScalingClient.builder()
               .region(Region.US_EAST_1)
               .build();

       System.out.println(DASHES);
       System.out.println("Welcome to the Amazon EC2 Auto Scaling example
scenario.");
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("1. Create an Auto Scaling group named " + groupName);
       createAutoScalingGroup(autoScalingClient, groupName, launchTemplateName,
vpcZoneId);
       System.out.println(
               "Wait 1 min for the resources, including the instance. Otherwise,
an empty instance Id is returned");
       Thread.sleep(60000);
       System.out.println(DASHES);

       System.out.println(DASHES);
```

```
        System.out.println("2. Get Auto Scale group Id value");
        String instanceId = getSpecificAutoScalingGroups(autoScalingClient,
groupName);
        if (instanceId.compareTo("") == 0) {
            System.out.println("Error - no instance Id value");
            System.exit(1);
        } else {
            System.out.println("The instance Id value is " + instanceId);
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Describe Auto Scaling with the Id value " +
instanceId);
        describeAutoScalingInstance(autoScalingClient, instanceId);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Enable metrics collection " + instanceId);
        enableMetricsCollection(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Update an Auto Scaling group to update max size to
3");
        updateAutoScalingGroup(autoScalingClient, groupName, launchTemplateName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Describe Auto Scaling groups");
        describeAutoScalingGroups(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Describe account details");
        describeAccountLimits(autoScalingClient);
        System.out.println(
                "Wait 1 min for the resources, including the instance. Otherwise,
an empty instance Id is returned");
        Thread.sleep(60000);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Set desired capacity to 2");
```

```
        setDesiredCapacity(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Get the two instance Id values and state");
        getSpecificAutoScalingGroups(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. List the scaling activities that have occurred
for the group");
        describeScalingActivities(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Terminate an instance in the Auto Scaling
group");
        terminateInstanceInAutoScalingGroup(autoScalingClient, instanceId);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Stop the metrics collection");
        disableMetricsCollection(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Delete the Auto Scaling group");
        deleteAutoScalingGroup(autoScalingClient, groupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);

        autoScalingClient.close();
    }

    public static void describeScalingActivities(AutoScalingClient
autoScalingClient, String groupName) {
        try {
            DescribeScalingActivitiesRequest scalingActivitiesRequest =
DescribeScalingActivitiesRequest.builder()
                    .autoScalingGroupName(groupName)
                    .maxRecords(10)
```

```
                    .build();

            DescribeScalingActivitiesResponse response = autoScalingClient
                    .describeScalingActivities(scalingActivitiesRequest);
            List<Activity> activities = response.activities();
            for (Activity activity : activities) {
                System.out.println("The activity Id is " +
activity.activityId());
                System.out.println("The activity details are " +
activity.details());
            }

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void setDesiredCapacity(AutoScalingClient autoScalingClient,
String groupName) {
        try {
            SetDesiredCapacityRequest capacityRequest =
SetDesiredCapacityRequest.builder()
                    .autoScalingGroupName(groupName)
                    .desiredCapacity(2)
                    .build();

            autoScalingClient.setDesiredCapacity(capacityRequest);
            System.out.println("You have set the DesiredCapacity to 2");

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createAutoScalingGroup(AutoScalingClient
autoScalingClient,
            String groupName,
            String launchTemplateName,
            String vpcZoneId) {
        try {
            AutoScalingWaiter waiter = autoScalingClient.waiter();
```

```
            LaunchTemplateSpecification templateSpecification =
LaunchTemplateSpecification.builder()
                    .launchTemplateName(launchTemplateName)
                    .build();

            CreateAutoScalingGroupRequest request =
CreateAutoScalingGroupRequest.builder()
                    .autoScalingGroupName(groupName)
                    .availabilityZones("us-east-1a")
                    .launchTemplate(templateSpecification)
                    .maxSize(1)
                    .minSize(1)
                    .vpcZoneIdentifier(vpcZoneId)
                    .build();

            autoScalingClient.createAutoScalingGroup(request);
            DescribeAutoScalingGroupsRequest groupsRequest =
DescribeAutoScalingGroupsRequest.builder()
                    .autoScalingGroupNames(groupName)
                    .build();

            WaiterResponse<DescribeAutoScalingGroupsResponse> waiterResponse =
waiter
                    .waitUntilGroupExists(groupsRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("Auto Scaling Group created");

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void describeAutoScalingInstance(AutoScalingClient
autoScalingClient, String id) {
        try {
            DescribeAutoScalingInstancesRequest
describeAutoScalingInstancesRequest = DescribeAutoScalingInstancesRequest
                    .builder()
                    .instanceIds(id)
                    .build();

            DescribeAutoScalingInstancesResponse response = autoScalingClient
```

```
.describeAutoScalingInstances(describeAutoScalingInstancesRequest);
           List<AutoScalingInstanceDetails> instances =
response.autoScalingInstances();
           for (AutoScalingInstanceDetails instance : instances) {
               System.out.println("The instance lifecycle state is: " +
instance.lifecycleState());
           }

       } catch (AutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void describeAutoScalingGroups(AutoScalingClient
autoScalingClient, String groupName) {
       try {
           DescribeAutoScalingGroupsRequest groupsRequest =
DescribeAutoScalingGroupsRequest.builder()
                   .autoScalingGroupNames(groupName)
                   .maxRecords(10)
                   .build();

           DescribeAutoScalingGroupsResponse response =
autoScalingClient.describeAutoScalingGroups(groupsRequest);
           List<AutoScalingGroup> groups = response.autoScalingGroups();
           for (AutoScalingGroup group : groups) {
               System.out.println("*** The service to use for the health checks:
" + group.healthCheckType());
           }

       } catch (AutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static String getSpecificAutoScalingGroups(AutoScalingClient
autoScalingClient, String groupName) {
       try {
           String instanceId = "";
           DescribeAutoScalingGroupsRequest scalingGroupsRequest =
DescribeAutoScalingGroupsRequest.builder()
```

```
                    .autoScalingGroupNames(groupName)
                    .build();

            DescribeAutoScalingGroupsResponse response = autoScalingClient
                    .describeAutoScalingGroups(scalingGroupsRequest);
            List<AutoScalingGroup> groups = response.autoScalingGroups();
            for (AutoScalingGroup group : groups) {
                System.out.println("The group name is " +
group.autoScalingGroupName());
                System.out.println("The group ARN is " +
group.autoScalingGroupARN());
                List<Instance> instances = group.instances();

                for (Instance instance : instances) {
                    instanceId = instance.instanceId();
                    System.out.println("The instance id is " + instanceId);
                    System.out.println("The lifecycle state is " +
instance.lifecycleState());
                }
            }

            return instanceId;
        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void enableMetricsCollection(AutoScalingClient
autoScalingClient, String groupName) {
        try {
            EnableMetricsCollectionRequest collectionRequest =
EnableMetricsCollectionRequest.builder()
                    .autoScalingGroupName(groupName)
                    .metrics("GroupMaxSize")
                    .granularity("1Minute")
                    .build();

            autoScalingClient.enableMetricsCollection(collectionRequest);
            System.out.println("The enable metrics collection operation was
successful");

        } catch (AutoScalingException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void disableMetricsCollection(AutoScalingClient
 autoScalingClient, String groupName) {
        try {
            DisableMetricsCollectionRequest disableMetricsCollectionRequest =
 DisableMetricsCollectionRequest.builder()
                    .autoScalingGroupName(groupName)
                    .metrics("GroupMaxSize")
                    .build();


 autoScalingClient.disableMetricsCollection(disableMetricsCollectionRequest);
            System.out.println("The disable metrics collection operation was
 successful");

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void describeAccountLimits(AutoScalingClient autoScalingClient)
 {
        try {
            DescribeAccountLimitsResponse response =
 autoScalingClient.describeAccountLimits();
            System.out.println("The max number of auto scaling groups is " +
 response.maxNumberOfAutoScalingGroups());
            System.out.println("The current number of auto scaling groups is " +
 response.numberOfAutoScalingGroups());

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void updateAutoScalingGroup(AutoScalingClient
 autoScalingClient, String groupName,
            String launchTemplateName) {
```

```
        try {
            AutoScalingWaiter waiter = autoScalingClient.waiter();
            LaunchTemplateSpecification templateSpecification =
LaunchTemplateSpecification.builder()
                    .launchTemplateName(launchTemplateName)
                    .build();

            UpdateAutoScalingGroupRequest groupRequest =
UpdateAutoScalingGroupRequest.builder()
                    .maxSize(3)
                    .autoScalingGroupName(groupName)
                    .launchTemplate(templateSpecification)
                    .build();

            autoScalingClient.updateAutoScalingGroup(groupRequest);
            DescribeAutoScalingGroupsRequest groupsRequest =
DescribeAutoScalingGroupsRequest.builder()
                    .autoScalingGroupNames(groupName)
                    .build();

            WaiterResponse<DescribeAutoScalingGroupsResponse> waiterResponse =
waiter
                    .waitUntilGroupInService(groupsRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("You successfully updated the auto scaling group
" + groupName);

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void terminateInstanceInAutoScalingGroup(AutoScalingClient
autoScalingClient, String instanceId) {
        try {
            TerminateInstanceInAutoScalingGroupRequest request =
TerminateInstanceInAutoScalingGroupRequest.builder()
                    .instanceId(instanceId)
                    .shouldDecrementDesiredCapacity(false)
                    .build();

            autoScalingClient.terminateInstanceInAutoScalingGroup(request);
            System.out.println("You have terminated instance " + instanceId);
```

```
        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteAutoScalingGroup(AutoScalingClient
 autoScalingClient, String groupName) {
        try {
            DeleteAutoScalingGroupRequest deleteAutoScalingGroupRequest =
 DeleteAutoScalingGroupRequest.builder()
                    .autoScalingGroupName(groupName)
                    .forceDelete(true)
                    .build();


 autoScalingClient.deleteAutoScalingGroup(deleteAutoScalingGroupRequest);
            System.out.println("You successfully deleted " + groupName);

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun main(args: Array<String>) {
    val usage = """
    Usage:
        <groupName> <launchTemplateName> <serviceLinkedRoleARN> <vpcZoneId>

    Where:
        groupName - The name of the Auto Scaling group.
        launchTemplateName - The name of the launch template.
        serviceLinkedRoleARN - The Amazon Resource Name (ARN) of the service-
linked role that the Auto Scaling group uses.
        vpcZoneId - A subnet Id for a virtual private cloud (VPC) where instances
 in the Auto Scaling group can be created.
    """

    if (args.size != 4) {
        println(usage)
        exitProcess(1)
    }

    val groupName = args[0]
    val launchTemplateName = args[1]
    val serviceLinkedRoleARN = args[2]
    val vpcZoneId = args[3]

    println("**** Create an Auto Scaling group named $groupName")
    createAutoScalingGroup(groupName, launchTemplateName, serviceLinkedRoleARN,
 vpcZoneId)

    println("Wait 1 min for the resources, including the instance. Otherwise, an
 empty instance Id is returned")
    delay(60000)
```

```
    val instanceId = getSpecificAutoScaling(groupName)
    if (instanceId.compareTo("") == 0) {
        println("Error - no instance Id value")
        exitProcess(1)
    } else {
        println("The instance Id value is $instanceId")
    }

    println("**** Describe Auto Scaling with the Id value $instanceId")
    describeAutoScalingInstance(instanceId)

    println("**** Enable metrics collection $instanceId")
    enableMetricsCollection(groupName)

    println("**** Update an Auto Scaling group to maximum size of 3")
    updateAutoScalingGroup(groupName, launchTemplateName, serviceLinkedRoleARN)

    println("**** Describe all Auto Scaling groups to show the current state of
 the groups")
    describeAutoScalingGroups(groupName)

    println("**** Describe account details")
    describeAccountLimits()

    println("Wait 1 min for the resources, including the instance. Otherwise, an
 empty instance Id is returned")
    delay(60000)

    println("**** Set desired capacity to 2")
    setDesiredCapacity(groupName)

    println("**** Get the two instance Id values and state")
    getAutoScalingGroups(groupName)

    println("**** List the scaling activities that have occurred for the group")
    describeScalingActivities(groupName)

    println("**** Terminate an instance in the Auto Scaling group")
    terminateInstanceInAutoScalingGroup(instanceId)

    println("**** Stop the metrics collection")
    disableMetricsCollection(groupName)

    println("**** Delete the Auto Scaling group")
```

```
    deleteSpecificAutoScalingGroup(groupName)
}

suspend fun describeAutoScalingGroups(groupName: String) {
    val groupsReques =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
            maxRecords = 10
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response = autoScalingClient.describeAutoScalingGroups(groupsReques)
        response.autoScalingGroups?.forEach { group ->
            println("The service to use for the health checks:
 ${group.healthCheckType}")
        }
    }
}

suspend fun disableMetricsCollection(groupName: String) {
    val disableMetricsCollectionRequest =
        DisableMetricsCollectionRequest {
            autoScalingGroupName = groupName
            metrics = listOf("GroupMaxSize")
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->

 autoScalingClient.disableMetricsCollection(disableMetricsCollectionRequest)
        println("The disable metrics collection operation was successful")
    }
}

suspend fun describeScalingActivities(groupName: String?) {
    val scalingActivitiesRequest =
        DescribeScalingActivitiesRequest {
            autoScalingGroupName = groupName
            maxRecords = 10
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeScalingActivities(scalingActivitiesRequest)
        response.activities?.forEach { activity ->
```

```
            println("The activity Id is ${activity.activityId}")
            println("The activity details are ${activity.details}")
        }
    }
}

suspend fun getAutoScalingGroups(groupName: String) {
    val scalingGroupsRequest =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeAutoScalingGroups(scalingGroupsRequest)
        response.autoScalingGroups?.forEach { group ->
            println("The group name is ${group.autoScalingGroupName}")
            println("The group ARN is ${group.autoScalingGroupArn}")
            group.instances?.forEach { instance ->
                println("The instance id is ${instance.instanceId}")
                println("The lifecycle state is " + instance.lifecycleState)
            }
        }
    }
}

suspend fun setDesiredCapacity(groupName: String) {
    val capacityRequest =
        SetDesiredCapacityRequest {
            autoScalingGroupName = groupName
            desiredCapacity = 2
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.setDesiredCapacity(capacityRequest)
        println("You set the DesiredCapacity to 2")
    }
}

suspend fun updateAutoScalingGroup(
    groupName: String,
    launchTemplateNameVal: String,
    serviceLinkedRoleARNVal: String,
) {
```

```
    val templateSpecification =
        LaunchTemplateSpecification {
            launchTemplateName = launchTemplateNameVal
        }

    val groupRequest =
        UpdateAutoScalingGroupRequest {
            maxSize = 3
            serviceLinkedRoleArn = serviceLinkedRoleARNVal
            autoScalingGroupName = groupName
            launchTemplate = templateSpecification
        }

    val groupsRequestWaiter =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.updateAutoScalingGroup(groupRequest)
        autoScalingClient.waitUntilGroupExists(groupsRequestWaiter)
        println("You successfully updated the Auto Scaling group  $groupName")
    }
}

suspend fun createAutoScalingGroup(
    groupName: String,
    launchTemplateNameVal: String,
    serviceLinkedRoleARNVal: String,
    vpcZoneIdVal: String,
) {
    val templateSpecification =
        LaunchTemplateSpecification {
            launchTemplateName = launchTemplateNameVal
        }

    val request =
        CreateAutoScalingGroupRequest {
            autoScalingGroupName = groupName
            availabilityZones = listOf("us-east-1a")
            launchTemplate = templateSpecification
            maxSize = 1
            minSize = 1
            vpcZoneIdentifier = vpcZoneIdVal
```

```
            serviceLinkedRoleArn = serviceLinkedRoleARNVal
        }

    // This object is required for the waiter call.
    val groupsRequestWaiter =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.createAutoScalingGroup(request)
        autoScalingClient.waitUntilGroupExists(groupsRequestWaiter)
        println("$groupName was created!")
    }
}

suspend fun describeAutoScalingInstance(id: String) {
    val describeAutoScalingInstancesRequest =
        DescribeAutoScalingInstancesRequest {
            instanceIds = listOf(id)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeAutoScalingInstances(describeAutoScalingInstancesRequest)
        response.autoScalingInstances?.forEach { group ->
            println("The instance lifecycle state is: ${group.lifecycleState}")
        }
    }
}

suspend fun enableMetricsCollection(groupName: String?) {
    val collectionRequest =
        EnableMetricsCollectionRequest {
            autoScalingGroupName = groupName
            metrics = listOf("GroupMaxSize")
            granularity = "1Minute"
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.enableMetricsCollection(collectionRequest)
        println("The enable metrics collection operation was successful")
    }
}
```

```
suspend fun getSpecificAutoScaling(groupName: String): String {
    var instanceId = ""
    val scalingGroupsRequest =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeAutoScalingGroups(scalingGroupsRequest)
        response.autoScalingGroups?.forEach { group ->
            println("The group name is ${group.autoScalingGroupName}")
            println("The group ARN is ${group.autoScalingGroupArn}")

            group.instances?.forEach { instance ->
                instanceId = instance.instanceId.toString()
            }
        }
    }
    return instanceId
}

suspend fun describeAccountLimits() {
    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeAccountLimits(DescribeAccountLimitsRequest {})
        println("The max number of Auto Scaling groups is
 ${response.maxNumberOfAutoScalingGroups}")
        println("The current number of Auto Scaling groups is
 ${response.numberOfAutoScalingGroups}")
    }
}

suspend fun terminateInstanceInAutoScalingGroup(instanceIdVal: String) {
    val request =
        TerminateInstanceInAutoScalingGroupRequest {
            instanceId = instanceIdVal
            shouldDecrementDesiredCapacity = false
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.terminateInstanceInAutoScalingGroup(request)
        println("You have terminated instance $instanceIdVal")
```

```
    }
}

suspend fun deleteSpecificAutoScalingGroup(groupName: String) {
    val deleteAutoScalingGroupRequest =
        DeleteAutoScalingGroupRequest {
            autoScalingGroupName = groupName
            forceDelete = true
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.deleteAutoScalingGroup(deleteAutoScalingGroupRequest)
        println("You successfully deleted $groupName")
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AutoScaling;

use Aws\AutoScaling\AutoScalingClient;
use Aws\CloudWatch\CloudWatchClient;
use Aws\Ec2\Ec2Client;
use AwsUtilities\AWSServiceClass;
use AwsUtilities\RunnableExample;

class GettingStartedWithAutoScaling implements RunnableExample
{
    protected Ec2Client $ec2Client;
    protected AutoScalingClient $autoScalingClient;
    protected AutoScalingService $autoScalingService;
    protected CloudWatchClient $cloudWatchClient;
    protected string $templateName;
    protected string $autoScalingGroupName;
    protected array $role;

    public function runExample()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon EC2 Auto Scaling getting started demo using
 PHP!\n");
        echo("--------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $this->autoScalingClient = new AutoScalingClient($clientArgs);
        $this->autoScalingService = new AutoScalingService($this-
>autoScalingClient);
        $this->cloudWatchClient = new CloudWatchClient($clientArgs);

        AWSServiceClass::$waitTime = 5;
        AWSServiceClass::$maxWaitAttempts = 20;

        /**
```

```
         * Step 0: Create an EC2 launch template that you'll use to create an
 Auto Scaling group.
         */
        $this->ec2Client = new EC2Client($clientArgs);
        $this->templateName = "example_launch_template_$uniqid";
        $instanceType = "t1.micro";
        $amiId = "ami-0ca285d4c2cda3300";
        $launchTemplate = $this->ec2Client->createLaunchTemplate(
            [
            'LaunchTemplateName' => $this->templateName,
            'LaunchTemplateData' => [
                'InstanceType' => $instanceType,
                'ImageId' => $amiId,
            ]
            ]
        );

        /**
         * Step 1: CreateAutoScalingGroup: pass it the launch template you
 created in step 0.
         */
        $availabilityZones[] = $this->ec2Client->describeAvailabilityZones([])
['AvailabilityZones'][1]['ZoneName'];

        $this->autoScalingGroupName = "demoAutoScalingGroupName_$uniqid";
        $minSize = 1;
        $maxSize = 1;
        $launchTemplateId = $launchTemplate['LaunchTemplate']
['LaunchTemplateId'];
        $this->autoScalingService->createAutoScalingGroup(
            $this->autoScalingGroupName,
            $availabilityZones,
            $minSize,
            $maxSize,
            $launchTemplateId
        );

        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);
        $autoScalingGroup = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);

        /**
```

```
         * Step 2: DescribeAutoScalingInstances: show that one instance has
 launched.
         */
        $instanceIds = [$autoScalingGroup['AutoScalingGroups'][0]['Instances'][0]
['InstanceId']];
        $instances = $this->autoScalingService-
>describeAutoScalingInstances($instanceIds);
        echo "The Auto Scaling group {$this->autoScalingGroupName} was created
 successfully.\n";
        echo count($instances['AutoScalingInstances']) . " instances were created
 for the group.\n";
        echo $autoScalingGroup['AutoScalingGroups'][0]['MaxSize'] . " is the max
 number of instances for the group.\n";

        /**
         * Step 3: EnableMetricsCollection: enable all metrics or a subset.
         */
        $this->autoScalingService->enableMetricsCollection($this-
>autoScalingGroupName, "1Minute");

        /**
         * Step 4: UpdateAutoScalingGroup: update max size to 3.
         */
        echo "Updating the max number of instances to 3.\n";
        $this->autoScalingService->updateAutoScalingGroup($this-
>autoScalingGroupName, ['MaxSize' => 3]);

        /**
         * Step 5: DescribeAutoScalingGroups: show the current state of the
 group.
         */
        $autoScalingGroup = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);
        echo $autoScalingGroup['AutoScalingGroups'][0]['MaxSize'];
        echo " is the updated max number of instances for the group.\n";

        $limits = $this->autoScalingService->describeAccountLimits();
        echo "Here are your account limits:\n";
        echo "MaxNumberOfAutoScalingGroups:
 {$limits['MaxNumberOfAutoScalingGroups']}\n";
        echo "MaxNumberOfLaunchConfigurations:
 {$limits['MaxNumberOfLaunchConfigurations']}\n";
        echo "NumberOfAutoScalingGroups:
 {$limits['NumberOfAutoScalingGroups']}\n";
```

```
        echo "NumberOfLaunchConfigurations:
 {$limits['NumberOfLaunchConfigurations']}\n";

        /**
         * Step 6: SetDesiredCapacity: set desired capacity to 2.
         */
        $this->autoScalingService->setDesiredCapacity($this-
>autoScalingGroupName, 2);
        sleep(10); // Wait for the group to start processing the request.
        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);

        /**
         * Step 7: DescribeAutoScalingInstances: show that two instances are
 launched.
         */
        $autoScalingGroups = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);
        foreach ($autoScalingGroups['AutoScalingGroups'] as $autoScalingGroup) {
            echo "There is a group named:
 {$autoScalingGroup['AutoScalingGroupName']}";
            echo "with an ARN of {$autoScalingGroup['AutoScalingGroupARN']}.\n";
            foreach ($autoScalingGroup['Instances'] as $instance) {
                echo "{$autoScalingGroup['AutoScalingGroupName']} has an instance
 with id of: ";
                echo "{$instance['InstanceId']} and a lifecycle state of:
 {$instance['LifecycleState']}.\n";
            }
        }

        /**
         * Step 8: TerminateInstanceInAutoScalingGroup: terminate one of the
 instances in the group.
         */
        $this->autoScalingService-
>terminateInstanceInAutoScalingGroup($instance['InstanceId'], false);
        do {
            sleep(10);
            $instances = $this->autoScalingService-
>describeAutoScalingInstances([$instance['InstanceId']]);
        } while (count($instances['AutoScalingInstances']) > 0);
        do {
            sleep(10);
```

```
            $autoScalingGroups = $this->autoScalingService-
>describeAutoScalingGroups([$this->autoScalingGroupName]);
            $instances = $autoScalingGroups['AutoScalingGroups'][0]['Instances'];
        } while (count($instances) < 2);
        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);
        foreach ($autoScalingGroups['AutoScalingGroups'] as $autoScalingGroup) {
            echo "There is a group named:
 {$autoScalingGroup['AutoScalingGroupName']}";
            echo "with an ARN of {$autoScalingGroup['AutoScalingGroupARN']}.\n";
            foreach ($autoScalingGroup['Instances'] as $instance) {
                echo "{$autoScalingGroup['AutoScalingGroupName']} has an instance
 with id of: ";
                echo "{$instance['InstanceId']} and a lifecycle state of:
 {$instance['LifecycleState']}.\n";
            }
        }

        /**
         * Step 9: DescribeScalingActivities: list the scaling activities that
 have occurred for the group so far.
         */
        $activities = $this->autoScalingService-
>describeScalingActivities($autoScalingGroup['AutoScalingGroupName']);
        echo "We found " . count($activities['Activities']) . " activities.\n";
        foreach ($activities['Activities'] as $activity) {
            echo "{$activity['ActivityId']} - {$activity['StartTime']} -
 {$activity['Description']}\n";
        }

        /**
         * Step 10: Use the Amazon CloudWatch API to get and show some metrics
 collected for the group.
         */
        $metricsNamespace = 'AWS/AutoScaling';
        $metricsDimensions = [
            [
                'Name' => 'AutoScalingGroupName',
                'Value' => $autoScalingGroup['AutoScalingGroupName'],
            ],
        ];
        $metrics = $this->cloudWatchClient->listMetrics(
            [
            'Dimensions' => $metricsDimensions,
```

```
            'Namespace' => $metricsNamespace,
            ]
        );
        foreach ($metrics['Metrics'] as $metric) {
            $timespan = 5;
            if ($metric['MetricName'] != 'GroupTotalCapacity' &&
 $metric['MetricName'] != 'GroupMaxSize') {
                continue;
            }
            echo "Over the last $timespan minutes, {$metric['MetricName']}
 recorded:\n";
            $stats = $this->cloudWatchClient->getMetricStatistics(
                [
                'Dimensions' => $metricsDimensions,
                'EndTime' => time(),
                'StartTime' => time() - (5 * 60),
                'MetricName' => $metric['MetricName'],
                'Namespace' => $metricsNamespace,
                'Period' => 60,
                'Statistics' => ['Sum'],
                ]
            );
            foreach ($stats['Datapoints'] as $stat) {
                echo "{$stat['Timestamp']}: {$stat['Sum']}\n";
            }
        }

        return $instances;
    }

    public function cleanUp()
    {
        /**
         * Step 11: DisableMetricsCollection: disable all metrics.
         */
        $this->autoScalingService->disableMetricsCollection($this-
>autoScalingGroupName);

        /**
         * Step 12: DeleteAutoScalingGroup: to delete the group you must stop all
 instances.
         * - UpdateAutoScalingGroup with MinSize=0
         * - TerminateInstanceInAutoScalingGroup for each instance,
```

```
         *     specify ShouldDecrementDesiredCapacity=True. Wait for instances to
 stop.
         * - Now you can delete the group.
         */
        $this->autoScalingService->updateAutoScalingGroup($this-
>autoScalingGroupName, ['MinSize' => 0]);
        $this->autoScalingService->terminateAllInstancesInAutoScalingGroup($this-
>autoScalingGroupName);
        $this->autoScalingService->waitUntilGroupInService([$this-
>autoScalingGroupName]);
        $this->autoScalingService->deleteAutoScalingGroup($this-
>autoScalingGroupName);

        /**
         * Step 13: Delete launch template.
         */
        $this->ec2Client->deleteLaunchTemplate(
            [
            'LaunchTemplateName' => $this->templateName,
            ]
        );
    }

    public function helloService()
    {
        $autoScalingClient = new AutoScalingClient([
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ]);

        $groups = $autoScalingClient->describeAutoScalingGroups([]);
        var_dump($groups);
    }
}
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances

- DescribeScalingActivities
- DisableMetricsCollection
- EnableMetricsCollection
- SetDesiredCapacity
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
def run_scenario(as_wrapper: AutoScalingWrapper, svc_helper: ServiceHelper) ->
 None:
    """
    Runs the scenario demonstrating the management of Auto Scaling groups and
 instances.

    :param as_wrapper: An instance of the AutoScalingWrapper that manages Auto
 Scaling groups.
    :param svc_helper: An instance of the ServiceHelper that interacts with AWS
 services.
    :return: None
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    logger.info("Starting the Amazon EC2 Auto Scaling demo.")

    print("-" * 88)
    print(
        "Welcome to the Amazon EC2 Auto Scaling demo for managing groups and
 instances."
```

```
    )
    print("-" * 88)

    print(
        "This example requires a launch template that specifies how to create "
        "EC2 instances. You can use an existing template or create a new one."
    )
    template_name = q.ask(
        "Enter the name of an existing launch template or press Enter to create a
new one: "
    )
    template = None
    if template_name:
        template = svc_helper.get_template(template_name)
    if template is None:
        inst_type = "t1.micro"
        ami_id = "ami-0ca285d4c2cda3300"
        print("Let's create a launch template with the following
specifications:")
        print(f"\tInstanceType: {inst_type}")
        print(f"\tAMI ID: {ami_id}")
        template_name = q.ask("Enter a name for the template: ", q.non_empty)
        template = svc_helper.create_template(template_name, inst_type, ami_id)
    print("-" * 88)

    print("Let's create an Auto Scaling group.")
    group_name = q.ask("Enter a name for the group: ", q.non_empty)
    zones = svc_helper.get_availability_zones()
    print("EC2 instances can be created in the following Availability Zones:")
    for index, zone in enumerate(zones):
        print(f"\t{index+1}. {zone}")
    print(f"\t{len(zones)+1}. All zones")
    zone_sel = q.ask(
        "Which zone do you want to use? ", q.is_int, q.in_range(1, len(zones) +
1)
    )
    group_zones = [zones[zone_sel - 1]] if zone_sel <= len(zones) else zones
    print(f"Creating group {group_name}...")
    as_wrapper.create_autoscaling_group(group_name, group_zones, template_name,
1, 1)
    wait(10)
    group = as_wrapper.describe_group(group_name)
    logger.info("Created Auto Scaling group %s.", group_name)
    print("Created group:")
```

```
    pp(group)
    print("Waiting for instance to start...")
    wait_for_group(group_name, as_wrapper)
    print("-" * 88)

    use_metrics = q.ask(
        "Do you want to collect metrics about Amazon EC2 Auto Scaling during this
 demo (y/n)? ",
        q.is_yesno,
    )
    if use_metrics:
        as_wrapper.enable_metrics(
            group_name,
            [
                "GroupMinSize",
                "GroupMaxSize",
                "GroupDesiredCapacity",
                "GroupInServiceInstances",
                "GroupTotalInstances",
            ],
        )
        logger.info("Enabled metrics for Auto Scaling group %s.", group_name)
        print(f"Metrics enabled for {group_name}.")
    print("-" * 88)

    print(f"Let's update the maximum number of instances in {group_name} from 1
 to 3.")
    q.ask("Press Enter when you're ready.")
    as_wrapper.update_group(group_name, MaxSize=3)
    group = as_wrapper.describe_group(group_name)
    logger.info("Updated maximum size for group %s to 3.", group_name)
    print("The group still has one running instance, but can have up to three:")
    print_simplified_group(group)
    print("-" * 88)

    print(f"Let's update the desired capacity of {group_name} from 1 to 2.")
    q.ask("Press Enter when you're ready.")
    as_wrapper.set_desired_capacity(group_name, 2)
    wait(10)
    group = as_wrapper.describe_group(group_name)
    logger.info("Set desired capacity for group %s to 2.", group_name)
    print("Here's the current state of the group:")
    print_simplified_group(group)
    print("-" * 88)
```

```
    print("Waiting for the new instance to start...")
    instance_ids = wait_for_group(group_name, as_wrapper)
    print("-" * 88)

    print(f"Let's terminate one of the instances in {group_name}.")
    print("Because the desired capacity is 2, another instance will start.")
    print("The currently running instances are:")
    for index, inst_id in enumerate(instance_ids):
        print(f"\t{index+1}. {inst_id}")
    inst_sel = q.ask(
        "Which instance do you want to stop? ",
        q.is_int,
        q.in_range(1, len(instance_ids) + 1),
    )
    print(f"Stopping {instance_ids[inst_sel-1]}...")
    as_wrapper.terminate_instance(instance_ids[inst_sel - 1], False)
    wait(10)
    group = as_wrapper.describe_group(group_name)
    logger.info(
        "Terminated instance %s in group %s.", instance_ids[inst_sel - 1],
group_name
    )
    print(f"Here's the state of {group_name}:")
    print_simplified_group(group)
    print("Waiting for the scaling activities to complete...")
    wait_for_group(group_name, as_wrapper)
    print("-" * 88)

    print(f"Let's get a report of scaling activities for {group_name}.")
    q.ask("Press Enter when you're ready.")
    activities = as_wrapper.describe_scaling_activities(group_name)
    logger.info(
        "Retrieved %d scaling activities for group %s.", len(activities),
group_name
    )
    print(
        f"Found {len(activities)} activities.\n"
        f"Activities are ordered with the most recent one first:"
    )
    for act in activities:
        pp(act)
    print("-" * 88)

    if use_metrics:
```

```
        print("Let's look at CloudWatch metrics.")
        metric_namespace = "AWS/AutoScaling"
        metric_dimensions = [{"Name": "AutoScalingGroupName", "Value":
group_name}]
        print(f"The following metrics are enabled for {group_name}:")
        done = False
        while not done:
            metrics = svc_helper.get_metrics(metric_namespace, metric_dimensions)
            for index, metric in enumerate(metrics):
                print(f"\t{index+1}. {metric.name}")
            print(f"\t{len(metrics)+1}. None")
            metric_sel = q.ask(
                "Which metric do you want to see? ",
                q.is_int,
                q.in_range(1, len(metrics) + 1),
            )
            if metric_sel < len(metrics) + 1:
                span = 5
                metric = metrics[metric_sel - 1]
                print(f"Over the last {span} minutes, {metric.name} recorded:")
                # CloudWatch metric times are in the UTC+0 time zone.
                now = datetime.now(timezone.utc)
                metric_data = svc_helper.get_metric_statistics(
                    metric_dimensions, metric, now - timedelta(minutes=span), now
                )
                pp(metric_data)
                if not q.ask("Do you want to see another metric (y/n)? ",
q.is_yesno):
                    done = True
            else:
                done = True

    print(f"Let's clean up.")
    q.ask("Press Enter when you're ready.")
    if use_metrics:
        print(f"Stopping metrics collection for {group_name}.")
        as_wrapper.disable_metrics(group_name)
        logger.info("Disabled metrics collection for group %s.", group_name)

    print(
        "You must terminate all instances in the group before you can delete the
group."
    )
    print("Set minimum size to 0.")
```

```
    as_wrapper.update_group(group_name, MinSize=0)
    group = as_wrapper.describe_group(group_name)
    instance_ids = [inst["InstanceId"] for inst in group["Instances"]]
    for inst_id in instance_ids:
        print(f"Stopping {inst_id}.")
        as_wrapper.terminate_instance(inst_id, True)
        logger.info("Terminated instance %s in group %s.", inst_id, group_name)
    print("Waiting for instances to stop...")
    wait_for_instances(instance_ids, as_wrapper)
    print(f"Deleting {group_name}.")
    as_wrapper.delete_autoscaling_group(group_name)
    logger.info("Deleted Auto Scaling group %s.", group_name)
    print("-" * 88)

    if template is not None:
        if q.ask(
            f"Do you want to delete launch template {template_name} used in this
 demo (y/n)? "
        ):
            svc_helper.delete_template(template_name)
            logger.info("Deleted launch template %s.", template_name)
            print("Template deleted.")

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        wrapper = AutoScalingWrapper(boto3.client("autoscaling"))
        helper = ServiceHelper(boto3.client("ec2"), boto3.resource("cloudwatch"))
        run_scenario(wrapper, helper)
    except Exception:
        logger.exception("Something went wrong with the demo!")
```

Define functions that are called by the scenario to manage launch templates and metrics. These functions wrap Amazon EC2 and CloudWatch actions.

```
class ServiceHelper:
    """Encapsulates Amazon EC2 and CloudWatch actions for the example."""

    def __init__(self, ec2_client, cloudwatch_resource):
```

```
        """
        :param ec2_client: A Boto3 Amazon EC2 client.
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.ec2_client = ec2_client
        self.cloudwatch_resource = cloudwatch_resource

    def get_template(self, template_name: str) -> dict:
        """
        Gets a launch template. Launch templates specify configuration for
instances
        that are launched by Amazon EC2 Auto Scaling.

        :param template_name: The name of the template to look up.
        :return: The template, if it exists.
        :raises ClientError: If there is an error retrieving the launch template.
        """
        try:
            response = self.ec2_client.describe_launch_templates(
                LaunchTemplateNames=[template_name]
            )
            template = response["LaunchTemplates"][0]
            logger.info("Launch template %s retrieved successfully.",
template_name)
            return template
        except ClientError as err:
            if (
                err.response["Error"]["Code"]
                == "InvalidLaunchTemplateName.NotFoundException"
            ):
                logger.warning("Launch template %s does not exist.",
template_name)
            else:
                logger.error(
                    "Couldn't verify launch template %s. Error: %s: %s",
                    template_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise

    def create_template(self, template_name: str, inst_type: str, ami_id: str) ->
dict:
        """
```

```
        Creates an Amazon EC2 launch template to use with Amazon EC2 Auto
Scaling.

        :param template_name: The name to give to the template.
        :param inst_type: The type of the instance, such as t1.micro.
        :param ami_id: The ID of the Amazon Machine Image (AMI) to use when
creating
                       an instance.
        :return: Information about the newly created template.
        :raises ClientError: If there is an error creating the launch template.
        """
        try:
            response = self.ec2_client.create_launch_template(
                LaunchTemplateName=template_name,
                LaunchTemplateData={"InstanceType": inst_type, "ImageId":
ami_id},
            )
            template = response["LaunchTemplate"]
            logger.info(
                "Created launch template %s with instance type %s and AMI ID
%s.",
                template_name,
                inst_type,
                ami_id,
            )
            return template
        except ClientError as err:
            logger.error(
                "Couldn't create launch template %s. Error: %s: %s",
                template_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

    def delete_template(self, template_name: str) -> None:
        """
        Deletes a launch template.

        :param template_name: The name of the template to delete.
        :raises ClientError: If there is an error deleting the launch template.
        """
        try:
```

```
self.ec2_client.delete_launch_template(LaunchTemplateName=template_name)
            logger.info("Deleted launch template %s.", template_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete launch template %s. Error: %s: %s",
                template_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

    def get_availability_zones(self) -> list:
        """
        Gets a list of Availability Zones in the AWS Region of the Amazon EC2
client.

        :return: The list of Availability Zones for the client Region.
        :raises ClientError: If there is an error retrieving availability zones.
        """
        try:
            response = self.ec2_client.describe_availability_zones()
            zones = [zone["ZoneName"] for zone in response["AvailabilityZones"]]
            logger.info("Retrieved availability zones: %s.", ", ".join(zones))
            return zones
        except ClientError as err:
            logger.error(
                "Couldn't get availability zones. Error: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

    def get_metrics(self, namespace: str, dimensions: list) -> list:
        """
        Gets a list of CloudWatch metrics filtered by namespace and dimensions.

        :param namespace: The namespace of the metrics to look up.
        :param dimensions: The dimensions of the metrics to look up.
        :return: The list of metrics.
        :raises ClientError: If there is an error retrieving CloudWatch metrics.
        """
        try:
            metrics = list(
```

```
                self.cloudwatch_resource.metrics.filter(
                    Namespace=namespace, Dimensions=dimensions
                )
            )
            logger.info(
                "Retrieved metrics for namespace %s with dimensions %s.",
                namespace,
                dimensions,
            )
            return metrics
        except ClientError as err:
            logger.error(
                "Couldn't get metrics for %s, %s. Error: %s: %s",
                namespace,
                dimensions,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

    @staticmethod
    def get_metric_statistics(
        dimensions: list, metric, start: datetime, end: datetime
    ) -> list:
        """
        Gets statistics for a CloudWatch metric within a specified time span.

        :param dimensions: The dimensions of the metric.
        :param metric: The metric to look up.
        :param start: The start of the time span for retrieved metrics.
        :param end: The end of the time span for retrieved metrics.
        :return: The list of data points found for the specified metric.
        :raises ClientError: If there is an error retrieving metric statistics.
        """
        try:
            response = metric.get_statistics(
                Dimensions=dimensions,
                StartTime=start,
                EndTime=end,
                Period=60,
                Statistics=["Sum"],
            )
            data = response["Datapoints"]
            logger.info("Retrieved statistics for metric %s.", metric.name)
```

```
            return data
        except ClientError as err:
            logger.error(
                "Couldn't get statistics for metric %s. Error: %s: %s",
                metric.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


def print_simplified_group(group: dict) -> None:
    """
    Prints a subset of data for an Auto Scaling group.

    :param group: The Auto Scaling group data to print.
    :return: None
    """
    print(group["AutoScalingGroupName"])
    print(f"\tLaunch template: {group['LaunchTemplate']['LaunchTemplateName']}")
    print(
        f"\tMin: {group['MinSize']}, Max: {group['MaxSize']}, Desired:
 {group['DesiredCapacity']}"
    )
    if group["Instances"]:
        print(f"\tInstances:")
        for inst in group["Instances"]:
            print(f"\t\t{inst['InstanceId']}: {inst['LifecycleState']}")


def wait_for_group(group_name: str, as_wrapper: AutoScalingWrapper) -> list:
    """
    Waits for instances to start or stop in an Auto Scaling group.
    Prints the data for each instance after scaling activities are complete.

    :param group_name: The name of the Auto Scaling group.
    :param as_wrapper: The AutoScalingWrapper that manages Auto Scaling groups.
    :return: A list of instance IDs in the group.
    """
    group = as_wrapper.describe_group(group_name)
    instance_ids = [i["InstanceId"] for i in group["Instances"]]
    return wait_for_instances(instance_ids, as_wrapper)
```

```
def wait_for_instances(instance_ids: list, as_wrapper: AutoScalingWrapper) ->
 list:
    """
    Waits for instances to start or stop in an Auto Scaling group.
    Prints the data for each instance after scaling activities are complete.

    :param instance_ids: A list of instance IDs to wait for.
    :param as_wrapper: The AutoScalingWrapper that manages Auto Scaling groups.
    :return: A list of instance IDs that were waited on.
    """
    ready = False
    instances = []
    while not ready:
        instances = as_wrapper.describe_instances(instance_ids) if instance_ids
 else []
        if all([x["LifecycleState"] in ["Terminated", "InService"] for x in
 instances]):
            ready = True
        else:
            wait(10)
    if instances:
        print(
            f"Here are the details of the instance{'s' if len(instances) > 1 else
 ''}:"
        )
        for instance in instances:
            pp(instance)
    return instance_ids
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection

- SetDesiredCapacity
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
[package]
name = "autoscaling-code-examples"
version = "0.1.0"
authors = ["Doug Schwartz <dougsch@amazon.com>", "David Souther
 <dpsouth@amazon.com>"]
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-autoscaling = { version = "1.3.0" }
aws-sdk-ec2 = { version = "1.3.0" }
aws-types = { version = "1.0.1" }
tokio = { version = "1.20.1", features = ["full"] }
clap = { version = "~4.4", features = ["derive"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
anyhow = "1.0.75"
tracing = "0.1.37"
tokio-stream = "0.1.14"


use std::{collections::BTreeSet, fmt::Display};

use anyhow::anyhow;
use autoscaling_code_examples::scenario::{AutoScalingScenario, ScenarioError};
```

```
use tracing::{info, warn};

async fn show_scenario_description(scenario: &AutoScalingScenario, event: &str) {
    let description = scenario.describe_scenario().await;
    info!("DescribeAutoScalingInstances: {event}\n{description}");
}

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    pub fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    pub fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();

    let shared_config = aws_config::from_env().load().await;

    let mut warnings = Warnings::default();

    // 1. Create an EC2 launch template that you'll use to create an auto scaling
 group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch
 template.
```

```
    // 2. CreateAutoScalingGroup: pass it the launch template you created in step
0. Give it min/max of 1 instance.
    // 4. EnableMetricsCollection: enable all metrics or a subset.
    let scenario = match
AutoScalingScenario::prepare_scenario(&shared_config).await {
        Ok(scenario) => scenario,
        Err(errs) => {
            let err_str = errs
                .into_iter()
                .map(|e| e.to_string())
                .collect::<Vec<String>>()
                .join(", ");
            return Err(anyhow!("Failed to initialize scenario: {err_str}"));
        }
    };

    info!("Prepared autoscaling scenario:\n{scenario}");

    let stable = scenario.wait_for_stable(1).await;
    if let Err(err) = stable {
        warnings.push(
            "There was a problem while waiting for group to be stable",
            err,
        );
    }

    // 3. DescribeAutoScalingInstances: show that one instance has launched.
    show_scenario_description(
        &scenario,
        "show that the group was created and one instance has launched",
    )
    .await;

    // 5. UpdateAutoScalingGroup: update max size to 3.
    let scale_max_size = scenario.scale_max_size(3).await;
    if let Err(err) = scale_max_size {
        warnings.push("There was a problem scaling max size", err);
    }

    // 6. DescribeAutoScalingGroups: the current state of the group
    show_scenario_description(
        &scenario,
        "show the current state of the group after setting max size",
    )
```

```
    .await;

    // 7. SetDesiredCapacity: set desired capacity to 2.
    let scale_desired_capacity = scenario.scale_desired_capacity(2).await;
    if let Err(err) = scale_desired_capacity {
        warnings.push("There was a problem setting desired capacity", err);
    }

    //   Wait for a second instance to launch.
    let stable = scenario.wait_for_stable(2).await;
    if let Err(err) = stable {
        warnings.push(
            "There was a problem while waiting for group to be stable",
            err,
        );
    }

    // 8. DescribeAutoScalingInstances: show that two instances are launched.
    show_scenario_description(
        &scenario,
        "show that two instances are launched after setting desired capacity",
    )
    .await;

    let ids_before = scenario
        .list_instances()
        .await
        .map(|v| v.into_iter().collect::<BTreeSet<_>>())
        .unwrap_or_default();

    // 9. TerminateInstanceInAutoScalingGroup: terminate one of the instances in
 the group.
    let terminate_some_instance = scenario.terminate_some_instance().await;
    if let Err(err) = terminate_some_instance {
        warnings.push("There was a problem replacing an instance", err);
    }

    let wait_after_terminate = scenario.wait_for_stable(1).await;
    if let Err(err) = wait_after_terminate {
        warnings.push(
            "There was a problem waiting after terminating an instance",
            err,
        );
    }
```

```
    let wait_scale_up_after_terminate = scenario.wait_for_stable(2).await;
    if let Err(err) = wait_scale_up_after_terminate {
        warnings.push(
            "There was a problem waiting for scale up after terminating an
instance",
            err,
        );
    }

    let ids_after = scenario
        .list_instances()
        .await
        .map(|v| v.into_iter().collect::<BTreeSet<_>>())
        .unwrap_or_default();

    let difference = ids_after.intersection(&ids_before).count();
    if !(difference == 1 && ids_before.len() == 2 && ids_after.len() == 2) {
        warnings.push(
            "Before and after set not different",
            ScenarioError::with(format!("{difference}")),
        );
    }

    // 10. DescribeScalingActivities: list the scaling activities that have
occurred for the group so far.
    show_scenario_description(
        &scenario,
        "list the scaling activities that have occurred for the group so far",
    )
    .await;

    // 11. DisableMetricsCollection
    let scale_group = scenario.scale_group_to_zero().await;
    if let Err(err) = scale_group {
        warnings.push("There was a problem scaling the group to 0", err);
    }
    show_scenario_description(&scenario, "Scenario scaled to 0").await;

    // 12. DeleteAutoScalingGroup (to delete the group you must stop all
instances):
    // 13. Delete LaunchTemplate.
    let clean_scenario = scenario.clean_scenario().await;
    if let Err(errs) = clean_scenario {
```

```
        for err in errs {
            warnings.push("There was a problem cleaning the scenario", err);
        }
    } else {
        info!("The scenario has been cleaned up!");
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        Err(anyhow!(
            "There were warnings during scenario execution:\n{warnings}"
        ))
    }
}

pub mod scenario;


use std::{
    error::Error,
    fmt::{Debug, Display},
    time::{Duration, SystemTime},
};

use anyhow::anyhow;
use aws_config::SdkConfig;
use aws_sdk_autoscaling::{
    error::{DisplayErrorContext, ProvideErrorMetadata},
    types::{Activity, AutoScalingGroup, LaunchTemplateSpecification},
};
use aws_sdk_ec2::types::RequestLaunchTemplateData;
use tracing::trace;

const LAUNCH_TEMPLATE_NAME: &str =
 "SDK_Code_Examples_EC2_Autoscaling_template_from_Rust_SDK";
const AUTOSCALING_GROUP_NAME: &str =
 "SDK_Code_Examples_EC2_Autoscaling_Group_from_Rust_SDK";
const MAX_WAIT: Duration = Duration::from_secs(5 * 60); // Wait at most 25
 seconds.
const WAIT_TIME: Duration = Duration::from_millis(500); // Wait half a second at
 a time.

struct Waiter {
```

```
    start: SystemTime,
    max: Duration,
}

impl Waiter {
    fn new() -> Self {
        Waiter {
            start: SystemTime::now(),
            max: MAX_WAIT,
        }
    }

    async fn sleep(&self) -> Result<(), ScenarioError> {
        if SystemTime::now()
            .duration_since(self.start)
            .unwrap_or(Duration::MAX)
            > self.max
        {
            Err(ScenarioError::with(
                "Exceeded maximum wait duration for stable group",
            ))
        } else {
            tokio::time::sleep(WAIT_TIME).await;
            Ok(())
        }
    }
}

pub struct AutoScalingScenario {
    ec2: aws_sdk_ec2::Client,
    autoscaling: aws_sdk_autoscaling::Client,
    launch_template_arn: String,
    auto_scaling_group_name: String,
}

impl Display for AutoScalingScenario {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        f.write_fmt(format_args!(
            "\tLaunch Template ID: {}\n",
            self.launch_template_arn
        ))?;
        f.write_fmt(format_args!(
            "\tScaling Group Name: {}\n",
            self.auto_scaling_group_name
```

```
        ))?;

        Ok(())
    }
}

pub struct AutoScalingScenarioDescription {
    group: Result<Vec<String>, ScenarioError>,
    instances: Result<Vec<String>, anyhow::Error>,
    activities: Result<Vec<Activity>, anyhow::Error>,
}

impl Display for AutoScalingScenarioDescription {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "\t      Group status:")?;
        match &self.group {
            Ok(groups) => {
                for status in groups {
                    writeln!(f, "\t\t- {status}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t! - {e}")?,
        }
        writeln!(f, "\t         Instances:")?;
        match &self.instances {
            Ok(instances) => {
                for instance in instances {
                    writeln!(f, "\t\t- {instance}")?;
                }
            }
            Err(e) => writeln!(f, "\t\t! {e}")?,
        }

        writeln!(f, "\t        Activities:")?;
        match &self.activities {
            Ok(activities) => {
                for activity in activities {
                    writeln!(
                        f,
                        "\t\t- {} Progress: {}% Status: {:?} End: {:?}",
                        activity.cause().unwrap_or("Unknown"),
                        activity.progress.unwrap_or(-1),
                        activity.status_code(),
                        // activity.status_message().unwrap_or_default()
```

```
                        activity.end_time(),
                    )?;
                }
            }
            Err(e) => writeln!(f, "\t\t! {e}")?,
        }

        Ok(())
    }
}

#[derive(Debug)]
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(|s| s.to_string()),
            code: err.code().map(|s| s.to_string()),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}
```

```
impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
    }
}

impl Error for ScenarioError {
    // While `Error` can capture `source` information about the underlying error,
 for this example
    // the ScenarioError captures the underlying information in MetadataError and
 treats it as a
    // single Error from this Crate. In other contexts, it may be appropriate to
 model the error
    // as including the SdkError as its source.
}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

impl AutoScalingScenario {
    pub async fn prepare_scenario(sdk_config: &SdkConfig) -> Result<Self,
 Vec<ScenarioError>> {
        let ec2 = aws_sdk_ec2::Client::new(sdk_config);
        let autoscaling = aws_sdk_autoscaling::Client::new(sdk_config);

        let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

        // Before creating any resources, prepare the list of AZs
```

```
        let availablity_zones = ec2.describe_availability_zones().send().await;
        if let Err(err) = availablity_zones {
            return Err(vec![ScenarioError::new("Failed to find AZs", &err)]);
        }

        let availability_zones: Vec<String> = availablity_zones
            .unwrap()
            .availability_zones
            .unwrap_or_default()
            .iter()
            .take(3)
            .map(|z| z.zone_name.clone().unwrap())
            .collect();

        // 1. Create an EC2 launch template that you'll use to create an auto
 scaling group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch
 template.
        //   * Recommended: InstanceType='t1.micro',
 ImageId='ami-0ca285d4c2cda3300'
        let create_launch_template = ec2
            .create_launch_template()
            .launch_template_name(LAUNCH_TEMPLATE_NAME)
            .launch_template_data(
                RequestLaunchTemplateData::builder()
                    .instance_type(aws_sdk_ec2::types::InstanceType::T1Micro)
                    .image_id("ami-0ca285d4c2cda3300")
                    .build(),
            )
            .send()
            .await
            .map_err(|err| vec![ScenarioError::new("Failed to create launch
 template", &err)])?;

        let launch_template_arn = match create_launch_template.launch_template {
            Some(launch_template) =>
 launch_template.launch_template_id.unwrap_or_default(),
            None => {
                // Try to delete the launch template
                let _ = ec2
                    .delete_launch_template()
                    .launch_template_name(LAUNCH_TEMPLATE_NAME)
                    .send()
                    .await;
```

```
                return Err(vec![ScenarioError::with("Failed to load launch
template")]);
            }
        };

        // 2. CreateAutoScalingGroup: pass it the launch template you created in
step 0. Give it min/max of 1 instance.
        //   You can use EC2.describe_availability_zones() to get a list of AZs
(you have to specify an AZ when you create the group).
        //   Wait for instance to launch. Use a waiter if you have one, otherwise
DescribeAutoScalingInstances until LifecycleState='InService'
        if let Err(err) = autoscaling
            .create_auto_scaling_group()
            .auto_scaling_group_name(auto_scaling_group_name.as_str())
            .launch_template(
                LaunchTemplateSpecification::builder()
                    .launch_template_id(launch_template_arn.clone())
                    .version("$Latest")
                    .build(),
            )
            .max_size(1)
            .min_size(1)
            .set_availability_zones(Some(availability_zones))
            .send()
            .await
        {
            let mut errs = vec![ScenarioError::new(
                "Failed to create autoscaling group",
                &err,
            )];

            if let Err(err) = autoscaling
                .delete_auto_scaling_group()
                .auto_scaling_group_name(auto_scaling_group_name.as_str())
                .send()
                .await
            {
                errs.push(ScenarioError::new(
                    "Failed to clean up autoscaling group",
                    &err,
                ));
            }

            if let Err(err) = ec2
```

```
                .delete_launch_template()
                .launch_template_id(launch_template_arn.clone())
                .send()
                .await
            {
                errs.push(ScenarioError::new(
                    "Failed to clean up launch template",
                    &err,
                ));
            }
            return Err(errs);
        }

        let scenario = AutoScalingScenario {
            ec2,
            autoscaling: autoscaling.clone(), // Clients are cheap so cloning
here to prevent a move is ok.
            auto_scaling_group_name: auto_scaling_group_name.clone(),
            launch_template_arn,
        };

        let enable_metrics_collection = autoscaling
            .enable_metrics_collection()
            .auto_scaling_group_name(auto_scaling_group_name.as_str())
            .granularity("1Minute")
            .set_metrics(Some(vec![
                String::from("GroupMinSize"),
                String::from("GroupMaxSize"),
                String::from("GroupDesiredCapacity"),
                String::from("GroupInServiceInstances"),
                String::from("GroupTotalInstances"),
            ]))
            .send()
            .await;

        match enable_metrics_collection {
            Ok(_) => Ok(scenario),
            Err(err) => {
                scenario.clean_scenario().await?;
                Err(vec![ScenarioError::new(
                    "Failed to enable metrics collections for group",
                    &err,
                )])
            }
```

```
        }
    }

    pub async fn clean_scenario(self) -> Result<(), Vec<ScenarioError>> {
        let _ = self.wait_for_no_scaling().await;
        let delete_group = self
            .autoscaling
            .delete_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .send()
            .await;

        // 14. Delete LaunchTemplate.
        let delete_launch_template = self
            .ec2
            .delete_launch_template()
            .launch_template_id(self.launch_template_arn.clone())
            .send()
            .await;

        let early_exit = match (delete_group, delete_launch_template) {
            (Ok(_), Ok(_)) => Ok(()),
            (Ok(_), Err(e)) => Err(vec![ScenarioError::new(
                "There was an error cleaning the launch template",
                &e,
            )]),
            (Err(e), Ok(_)) => Err(vec![ScenarioError::new(
                "There was an error cleaning the scale group",
                &e,
            )]),
            (Err(e1), Err(e2)) => Err(vec![
                ScenarioError::new("Multiple error cleaning the scenario Scale
Group", &e1),
                ScenarioError::new("Multiple error cleaning the scenario Launch
Template", &e2),
            ]),
        };

        if early_exit.is_err() {
            early_exit
        } else {
            // Wait for delete_group to finish
            let waiter = Waiter::new();
            let mut errors = Vec::<ScenarioError>::new();
```

```
            while errors.len() < 3 {
                if let Err(e) = waiter.sleep().await {
                    errors.push(e);
                    continue;
                }
                let describe_group = self
                    .autoscaling
                    .describe_auto_scaling_groups()

.auto_scaling_group_names(self.auto_scaling_group_name.clone())
                    .send()
                    .await;
                match describe_group {
                    Ok(group) => match group.auto_scaling_groups().first() {
                        Some(group) => {
                            if group.status() != Some("Delete in progress") {
                                errors.push(ScenarioError::with(format!(
                                    "Group in an unknown state while deleting:
{}",
                                    group.status().unwrap_or("unknown error")
                                )));
                                return Err(errors);
                            }
                        }
                        None => return Ok(()),
                    },
                    Err(err) => {
                        errors.push(ScenarioError::new("Failed to describe
autoscaling group during cleanup 3 times, last error", &err));
                    }
                }
                if errors.len() > 3 {
                    return Err(errors);
                }
            }
            Err(vec![ScenarioError::with(
                "Exited cleanup wait loop without retuning success or failing
after three rounds",
            )])
        }
    }

    pub async fn describe_scenario(&self) -> AutoScalingScenarioDescription {
        let group = self
```

```
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await
            .map(|s| {
                s.auto_scaling_groups()
                    .iter()
                    .map(|s| {
                        format!(
                            "{}: {}",
                            s.auto_scaling_group_name().unwrap_or("Unknown"),
                            s.status().unwrap_or("Unknown")
                        )
                    })
                    .collect::<Vec<String>>()
            })
            .map_err(|e| {
                ScenarioError::new("Failed to describe auto scaling groups for
scenario", &e)
            });

        let instances = self
            .list_instances()
            .await
            .map_err(|e| anyhow!("There was an error listing instances: {e}",));

        // 10. DescribeScalingActivities: list the scaling activities that have
occurred for the group so far.
        //   Bonus: use CloudWatch API to get and show some metrics collected for
the group.
        //   CW.ListMetrics with Namespace='AWS/AutoScaling' and
Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
        //   CW.GetMetricStatistics with Statistics='Sum'. Start and End times
must be in UTC!
        let activities = self
            .autoscaling
            .describe_scaling_activities()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .into_paginator()
            .items()
            .send()
            .collect::<Result<Vec<_>, _>>()
            .await
```

```
            .map_err(|e| {
                anyhow!(
                    "There was an error retrieving scaling activities: {}",
                    DisplayErrorContext(&e)
                )
            });

        AutoScalingScenarioDescription {
            group,
            instances,
            activities,
        }
    }

    async fn get_group(&self) -> Result<AutoScalingGroup, ScenarioError> {
        let describe_auto_scaling_groups = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await;

        if let Err(err) = describe_auto_scaling_groups {
            return Err(ScenarioError::new(
                format!(
                    "Failed to get status of autoscaling group {}",
                    self.auto_scaling_group_name.clone()
                )
                .as_str(),
                &err,
            ));
        }

        let describe_auto_scaling_groups_output =
describe_auto_scaling_groups.unwrap();
        let auto_scaling_groups =
describe_auto_scaling_groups_output.auto_scaling_groups();
        let auto_scaling_group = auto_scaling_groups.first();

        if auto_scaling_group.is_none() {
            return Err(ScenarioError::with(format!(
                "Could not find autoscaling group {}",
                self.auto_scaling_group_name.clone()
            )));
```

```
        }

        Ok(auto_scaling_group.unwrap().clone())
    }

    pub async fn wait_for_no_scaling(&self) -> Result<(), ScenarioError> {
        let waiter = Waiter::new();
        let mut scaling = true;
        while scaling {
            waiter.sleep().await?;
            let describe_activities = self
                .autoscaling
                .describe_scaling_activities()
                .auto_scaling_group_name(self.auto_scaling_group_name.clone())
                .send()
                .await
                .map_err(|e| {
                    ScenarioError::new("Failed to get autoscaling activities for
group", &e)
                })?;
            let activities = describe_activities.activities();
            trace!(
                "Waiting for no scaling found {} activities",
                activities.len()
            );
            scaling = activities.iter().any(|a| a.progress() < Some(100));
        }
        Ok(())
    }

    pub async fn wait_for_stable(&self, size: usize) -> Result<(), ScenarioError>
{
        self.wait_for_no_scaling().await?;

        let mut group = self.get_group().await?;
        let mut count = count_group_instances(&group);

        let waiter = Waiter::new();
        while count != size {
            trace!("Waiting for stable {size} (current: {count})");
            waiter.sleep().await?;
            group = self.get_group().await?;
            count = count_group_instances(&group);
        }
```

```
        Ok(())
    }

    pub async fn list_instances(&self) -> Result<Vec<String>, ScenarioError> {
        // The direct way to list instances is by using
 DescribeAutoScalingGroup's instances property. However, this returns a
 Vec<Instance>, as opposed to a Vec<AutoScalingInstanceDetails>.
        // Ok(self.get_group().await?.instances.unwrap_or_default().map(|
i| i.instance_id.clone().unwrap_or_default()).filter(|id| !
id.is_empty()).collect())

        // Alternatively, and for the sake of example,
 DescribeAutoScalingInstances returns a list that can be filtered by the client.
        self.autoscaling
            .describe_auto_scaling_instances()
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
            .map(|items| {
                items
                    .into_iter()
                    .filter(|i| {
                        i.auto_scaling_group_name.as_deref()
                            == Some(self.auto_scaling_group_name.as_str())
                    })
                    .map(|i| i.instance_id.unwrap_or_default())
                    .filter(|id| !id.is_empty())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Failed to get list of auto scaling
 instances", &err))
    }

    pub async fn scale_min_size(&self, size: i32) -> Result<(), ScenarioError> {
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .min_size(size)
            .send()
            .await;
```

```
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failer to update group to min size ({size}))").as_str(),
                &err,
            ));
        }
        Ok(())
    }

    pub async fn scale_max_size(&self, size: i32) -> Result<(), ScenarioError> {
        // 5. UpdateAutoScalingGroup: update max size to 3.
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .max_size(size)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failed to update group to max size ({size})").as_str(),
                &err,
            ));
        }
        Ok(())
    }

    pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
 ScenarioError> {
        // 7. SetDesiredCapacity: set desired capacity to 2.
        //   Wait for a second instance to launch.
        let update_group = self
            .autoscaling
            .set_desired_capacity()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .desired_capacity(capacity)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failed to update group to desired capacity
 ({capacity}))").as_str(),
                &err,
            ));
```

```
        }
        Ok(())
    }

    pub async fn scale_group_to_zero(&self) -> Result<(), ScenarioError> {
        // If this fails it's fine, just means there are extra cloudwatch metrics
events for the scale-down.
        let _ = self
            .autoscaling
            .disable_metrics_collection()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .send()
            .await;

        // 12. DeleteAutoScalingGroup (to delete the group you must stop all
instances):
        //   UpdateAutoScalingGroup with MinSize=0
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .min_size(0)
            .desired_capacity(0)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                "Failed to update group for scaling down&",
                &err,
            ));
        }

        let stable = self.wait_for_stable(0).await;
        if let Err(err) = stable {
            return Err(ScenarioError::with(format!(
                "Error while waiting for group to be stable on scale down: {err}"
            )));
        }

        Ok(())
    }

    pub async fn terminate_some_instance(&self) -> Result<(), ScenarioError> {
        // Retrieve a list of instances in the auto scaling group.
```

```
        let auto_scaling_group = self.get_group().await?;
        let instances = auto_scaling_group.instances();
        // Or use other logic to find an instance to terminate.
        let instance = instances.first();
        if let Some(instance) = instance {
            let instance_id = if let Some(instance_id) = instance.instance_id() {
                instance_id
            } else {
                return Err(ScenarioError::with("Missing instance id"));
            };
            let termination = self
                .ec2
                .terminate_instances()
                .instance_ids(instance_id)
                .send()
                .await;
            if let Err(err) = termination {
                Err(ScenarioError::new(
                    "There was a problem terminating an instance",
                    &err,
                ))
            } else {
                Ok(())
            }
        } else {
            Err(ScenarioError::with("There was no instance to terminate"))
        }
    }
}

fn count_group_instances(group: &AutoScalingGroup) -> usize {
    group.instances.as_ref().map(|i| i.len()).unwrap_or(0)
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities

- DisableMetricsCollection
- EnableMetricsCollection
- SetDesiredCapacity
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for Auto Scaling using AWS SDKs

The following code examples demonstrate how to perform individual Auto Scaling actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Auto Scaling API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Auto Scaling using AWS SDKs .

The following examples include only the most commonly used actions. For a complete list, see the Amazon EC2 Auto Scaling API Reference.

Examples

- Use AttachInstances with an AWS SDK or CLI
- Use AttachLoadBalancerTargetGroups with an AWS SDK or CLI
- Use AttachLoadBalancers with an AWS SDK or CLI
- Use CompleteLifecycleAction with an AWS SDK or CLI
- Use CreateAutoScalingGroup with an AWS SDK or CLI
- Use CreateLaunchConfiguration with an AWS SDK or CLI
- Use CreateOrUpdateTags with an AWS SDK or CLI
- Use DeleteAutoScalingGroup with an AWS SDK or CLI
- Use DeleteLaunchConfiguration with an AWS SDK or CLI
- Use DeleteLifecycleHook with an AWS SDK or CLI
- Use DeleteNotificationConfiguration with an AWS SDK or CLI
- Use DeletePolicy with an AWS SDK or CLI

- Use DeleteScheduledAction with an AWS SDK or CLI
- Use DeleteTags with an AWS SDK or CLI
- Use DescribeAccountLimits with an AWS SDK or CLI
- Use DescribeAdjustmentTypes with an AWS SDK or CLI
- Use DescribeAutoScalingGroups with an AWS SDK or CLI
- Use DescribeAutoScalingInstances with an AWS SDK or CLI
- Use DescribeAutoScalingNotificationTypes with an AWS SDK or CLI
- Use DescribeLaunchConfigurations with an AWS SDK or CLI
- Use DescribeLifecycleHookTypes with an AWS SDK or CLI
- Use DescribeLifecycleHooks with an AWS SDK or CLI
- Use DescribeLoadBalancers with an AWS SDK or CLI
- Use DescribeMetricCollectionTypes with an AWS SDK or CLI
- Use DescribeNotificationConfigurations with an AWS SDK or CLI
- Use DescribePolicies with an AWS SDK or CLI
- Use DescribeScalingActivities with an AWS SDK or CLI
- Use DescribeScalingProcessTypes with an AWS SDK or CLI
- Use DescribeScheduledActions with an AWS SDK or CLI
- Use DescribeTags with an AWS SDK or CLI
- Use DescribeTerminationPolicyTypes with an AWS SDK or CLI
- Use DetachInstances with an AWS SDK or CLI
- Use DetachLoadBalancers with an AWS SDK or CLI
- Use DisableMetricsCollection with an AWS SDK or CLI
- Use EnableMetricsCollection with an AWS SDK or CLI
- Use EnterStandby with an AWS SDK or CLI
- Use ExecutePolicy with an AWS SDK or CLI
- Use ExitStandby with an AWS SDK or CLI
- Use PutLifecycleHook with an AWS SDK or CLI
- Use PutNotificationConfiguration with an AWS SDK or CLI
- Use PutScalingPolicy with an AWS SDK or CLI
- Use PutScheduledUpdateGroupAction with an AWS SDK or CLI

- Use RecordLifecycleActionHeartbeat with an AWS SDK or CLI
- Use ResumeProcesses with an AWS SDK or CLI
- Use SetDesiredCapacity with an AWS SDK or CLI
- Use SetInstanceHealth with an AWS SDK or CLI
- Use SetInstanceProtection with an AWS SDK or CLI
- Use SuspendProcesses with an AWS SDK or CLI
- Use TerminateInstanceInAutoScalingGroup with an AWS SDK or CLI
- Use UpdateAutoScalingGroup with an AWS SDK or CLI

Use `AttachInstances` with an AWS SDK or CLI

The following code examples show how to use `AttachInstances`.

CLI

AWS CLI

To attach an instance to an Auto Scaling group

This example attaches the specified instance to the specified Auto Scaling group.

```
aws autoscaling attach-instances \
    --instance-ids i-061c63c5eb45f0416 \
    --auto-scaling-group-name my-asg
```

This command produces no output.

- For API details, see AttachInstances in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example attaches the specified instance to the specified Auto Scaling group. Auto Scaling automatically increases the desired capacity of the Auto Scaling group.

```
Mount-ASInstance -InstanceId i-93633f9b -AutoScalingGroupName my-asg
```

- For API details, see AttachInstances in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AttachLoadBalancerTargetGroups` with an AWS SDK or CLI

The following code examples show how to use `AttachLoadBalancerTargetGroups`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Build and manage a resilient service

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName,
 string targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
```

```
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
```

- For API details, see AttachLoadBalancerTargetGroups in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To attach a target group to an Auto Scaling group

This example attaches the specified target group to the specified Auto Scaling group.

```
aws autoscaling attach-load-balancer-target-groups \
    --auto-scaling-group-name my-asg \
    --target-group-arns arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/my-targets/73e2d6bc24d8a067
```

This command produces no output.

For more information, see Elastic Load Balancing and Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see AttachLoadBalancerTargetGroups in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    const client = new AutoScalingClient({});
    await client.send(
```

```
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
```

- For API details, see AttachLoadBalancerTargetGroups in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """
    Encapsulates Amazon EC2 Auto Scaling and EC2 management actions.
    """

    def __init__(
        self,
        resource_prefix: str,
        inst_type: str,
        ami_param: str,
        autoscaling_client: boto3.client,
        ec2_client: boto3.client,
        ssm_client: boto3.client,
        iam_client: boto3.client,
    ):
        """
        Initializes the AutoScaler class with the necessary parameters.

        :param resource_prefix: The prefix for naming AWS resources that are
 created by this class.
        :param inst_type: The type of EC2 instance to create, such as t3.micro.
```

```
        :param ami_param: The Systems Manager parameter used to look up the AMI
that is created.
        :param autoscaling_client: A Boto3 EC2 Auto Scaling client.
        :param ec2_client: A Boto3 EC2 client.
        :param ssm_client: A Boto3 Systems Manager client.
        :param iam_client: A Boto3 IAM client.
        """
        self.inst_type = inst_type
        self.ami_param = ami_param
        self.autoscaling_client = autoscaling_client
        self.ec2_client = ec2_client
        self.ssm_client = ssm_client
        self.iam_client = iam_client
        sts_client = boto3.client("sts")
        self.account_id = sts_client.get_caller_identity()["Account"]

        self.key_pair_name = f"{resource_prefix}-key-pair"
        self.launch_template_name = f"{resource_prefix}-template-"
        self.group_name = f"{resource_prefix}-group"

        # Happy path
        self.instance_policy_name = f"{resource_prefix}-pol"
        self.instance_role_name = f"{resource_prefix}-role"
        self.instance_profile_name = f"{resource_prefix}-prof"

        # Failure mode
        self.bad_creds_policy_name = f"{resource_prefix}-bc-pol"
        self.bad_creds_role_name = f"{resource_prefix}-bc-role"
        self.bad_creds_profile_name = f"{resource_prefix}-bc-prof"


    def attach_load_balancer_target_group(
        self, lb_target_group: Dict[str, Any]
    ) -> None:
        """
        Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
        The target group specifies how the load balancer forwards requests to the
instances
        in the group.

        :param lb_target_group: Data about the ELB target group to attach.
        """
        try:
```

```
            self.autoscaling_client.attach_load_balancer_target_groups(
                AutoScalingGroupName=self.group_name,
                TargetGroupARNs=[lb_target_group["TargetGroupArn"]],
            )
            log.info(
                "Attached load balancer target group %s to auto scaling group
%s.",
                lb_target_group["TargetGroupName"],
                self.group_name,
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to attach load balancer target group
'{lb_target_group['TargetGroupName']}'."
            )
            if error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
the resource."
                )
            elif error_code == "ServiceLinkedRoleFailure":
                log.error(
                    "The operation failed because the service-linked role is not
ready or does not exist. "
                    "Check that the service-linked role exists and is correctly
configured."
                )
            log.error(f"Full error:\n\t{err}")
```

- For API details, see AttachLoadBalancerTargetGroups in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AttachLoadBalancers` with an AWS SDK or CLI

The following code examples show how to use `AttachLoadBalancers`.

CLI

AWS CLI

To attach a Classic Load Balancer to an Auto Scaling group

This example attaches the specified Classic Load Balancer to the specified Auto Scaling group.

```
aws autoscaling attach-load-balancers \
    --load-balancer-names my-load-balancer \
    --auto-scaling-group-name my-asg
```

This command produces no output.

For more information, see Elastic Load Balancing and Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see AttachLoadBalancers in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example attaches the specified load balancer to the specified Auto Scaling group.

```
Add-ASLoadBalancer -LoadBalancerName my-lb -AutoScalingGroupName my-asg
```

- For API details, see AttachLoadBalancers in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CompleteLifecycleAction` with an AWS SDK or CLI

The following code examples show how to use `CompleteLifecycleAction`.

CLI

AWS CLI

To complete the lifecycle action

This example notifies Amazon EC2 Auto Scaling that the specified lifecycle action is complete so that it can finish launching or terminating the instance.

```
aws autoscaling complete-lifecycle-action \
    --lifecycle-hook-name my-launch-hook \
    --auto-scaling-group-name my-asg \
    --lifecycle-action-result CONTINUE \
    --lifecycle-action-token bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635
```

This command produces no output.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see CompleteLifecycleAction in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example completes the specified lifecycle action.

```
Complete-ASLifecycleAction -LifecycleHookName myLifecycleHook -
AutoScalingGroupName my-asg -LifecycleActionResult CONTINUE -LifecycleActionToken
 bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635
```

- For API details, see CompleteLifecycleAction in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateAutoScalingGroup` with an AWS SDK or CLI

The following code examples show how to use `CreateAutoScalingGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Build and manage a resilient service

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
    string availabilityZone)
{
    var templateSpecification = new LaunchTemplateSpecification
    {
        LaunchTemplateName = launchTemplateName,
    };

    var zoneList = new List<string>
        {
```

```
                availabilityZone,
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };

        var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::CreateAutoScalingGroupRequest request;
        request.SetAutoScalingGroupName(groupName);
        Aws::Vector<Aws::String> availabilityGroupZones;
        availabilityGroupZones.push_back(
```

```
                availabilityZones[availabilityZoneChoice - 1].GetZoneName());
        request.SetAvailabilityZones(availabilityGroupZones);
        request.SetMaxSize(1);
        request.SetMinSize(1);

        Aws::AutoScaling::Model::LaunchTemplateSpecification
launchTemplateSpecification;
        launchTemplateSpecification.SetLaunchTemplateName(templateName);
        request.SetLaunchTemplate(launchTemplateSpecification);

        Aws::AutoScaling::Model::CreateAutoScalingGroupOutcome outcome =
                autoScalingClient.CreateAutoScalingGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "Created Auto Scaling group '" << groupName << "'..."
                      << std::endl;
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::AutoScaling::AutoScalingErrors::ALREADY_EXISTS_FAULT) {
            std::cout << "Auto Scaling group '" << groupName << "' already
exists."
                      << std::endl;
        }
        else {
            std::cerr << "Error with AutoScaling::CreateAutoScalingGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

        }
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To create an Auto Scaling group

The following `create-auto-scaling-group` example creates an Auto Scaling group in subnets in multiple Availability Zones within a Region. The instances launch with the

default version of the specified launch template. Note that defaults are used for most other settings, such as the termination policies and health check configuration.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-template LaunchTemplateId=lt-1234567890abcde12 \
    --min-size 1 \
    --max-size 5 \
    --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
```

This command produces no output.

For more information, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To attach an Application Load Balancer, Network Load Balancer, or Gateway Load Balancer

This example specifies the ARN of a target group for a load balancer that supports the expected traffic. The health check type specifies `ELB` so that when Elastic Load Balancing reports an instance as unhealthy, the Auto Scaling group replaces it. The command also defines a health check grace period of `600` seconds. The grace period helps prevent premature termination of newly launched instances.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-template LaunchTemplateId=lt-1234567890abcde12 \
    --target-group-arns arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/my-targets/943f017f100becff \
    --health-check-type ELB \
    --health-check-grace-period 600 \
    --min-size 1 \
    --max-size 5 \
    --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
```

This command produces no output.

For more information, see Elastic Load Balancing and Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To specify a placement group and use the latest version of the launch template

This example launches instances into a placement group within a single Availability Zone. This can be useful for low-latency groups with HPC workloads. This example also specifies the minimum size, maximum size, and desired capacity of the group.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-template LaunchTemplateId=lt-1234567890abcde12,Version='$Latest' \
    --min-size 1 \
    --max-size 5 \
    --desired-capacity 3 \
    --placement-group my-placement-group \
    --vpc-zone-identifier "subnet-6194ea3b"
```

This command produces no output.

For more information, see Placement groups in the *Amazon EC2 User Guide for Linux Instances*.

Example 4: To specify a single instance Auto Scaling group and use a specific version of the launch template

This example creates an Auto Scaling group with minimum and maximum capacity set to 1 to enforce that one instance will be running. The command also specifies v1 of a launch template in which the ID of an existing ENI is specified. When you use a launch template that specifies an existing ENI for eth0, you must specify an Availability Zone for the Auto Scaling group that matches the network interface, without also specifying a subnet ID in the request.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg-single-instance \
    --launch-template LaunchTemplateName=my-template-for-auto-scaling,Version='1'
 \
    --min-size 1 \
    --max-size 1 \
    --availability-zones us-west-2a
```

This command produces no output.

For more information, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.

Example 5: To specify a different termination policy

This example creates an Auto Scaling group using a launch configuration and sets the termination policy to terminate the oldest instances first. The command also applies a tag to the group and its instances, with a key of `Role` and a value of `WebServer`.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-configuration-name my-lc \
    --min-size 1 \
    --max-size 5 \
    --termination-policies "OldestInstance" \
    --tags "ResourceId=my-asg,ResourceType=auto-scaling-
group,Key=Role,Value=WebServer,PropagateAtLaunch=true" \
    --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
```

This command produces no output.

For more information, see Working with Amazon EC2 Auto Scaling termination policies in the *Amazon EC2 Auto Scaling User Guide*.

Example 6: To specify a launch lifecycle hook

This example creates an Auto Scaling group with a lifecycle hook that supports a custom action at instance launch.

```
aws autoscaling create-auto-scaling-group \
    --cli-input-json file://~/config.json
```

Contents of `config.json` file:

```
{
    "AutoScalingGroupName": "my-asg",
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-1234567890abcde12"
    },
    "LifecycleHookSpecificationList": [{
        "LifecycleHookName": "my-launch-hook",
        "LifecycleTransition": "autoscaling:EC2_INSTANCE_LAUNCHING",
        "NotificationTargetARN": "arn:aws:sqs:us-west-2:123456789012:my-sqs-
queue",
        "RoleARN": "arn:aws:iam::123456789012:role/my-notification-role",
        "NotificationMetadata": "SQS message metadata",
        "HeartbeatTimeout": 4800,
```

```
        "DefaultResult": "ABANDON"
    }],
    "MinSize": 1,
    "MaxSize": 5,
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782",
    "Tags": [{
        "ResourceType": "auto-scaling-group",
        "ResourceId": "my-asg",
        "PropagateAtLaunch": true,
        "Value": "test",
        "Key": "environment"
    }]
}
```

This command produces no output.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

Example 7: To specify a termination lifecycle hook

This example creates an Auto Scaling group with a lifecycle hook that supports a custom action at instance termination.

```
aws autoscaling create-auto-scaling-group \
    --cli-input-json file://~/config.json
```

Contents of `config.json`:

```
{
    "AutoScalingGroupName": "my-asg",
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-1234567890abcde12"
    },
    "LifecycleHookSpecificationList": [{
        "LifecycleHookName": "my-termination-hook",
        "LifecycleTransition": "autoscaling:EC2_INSTANCE_TERMINATING",
        "HeartbeatTimeout": 120,
        "DefaultResult": "CONTINUE"
    }],
    "MinSize": 1,
    "MaxSize": 5,
    "TargetGroupARNs": [
```

```
        "arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/my-
targets/73e2d6bc24d8a067"
    ],
    "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
}
```

This command produces no output.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

Example 8: To specify a custom termination policy

This example creates an Auto Scaling group that specifies a custom Lambda function termination policy that tells Amazon EC2 Auto Scaling which instances are safe to terminate on scale in.

```
aws autoscaling create-auto-scaling-group \
    --auto-scaling-group-name my-asg-single-instance \
    --launch-template LaunchTemplateName=my-template-for-auto-scaling \
    --min-size 1 \
    --max-size 5 \
    --termination-policies "arn:aws:lambda:us-
west-2:123456789012:function:HelloFunction:prod" \
    --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
```

This command produces no output.

For more information, see Creating a custom termination policy with Lambda in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see CreateAutoScalingGroup in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.autoscaling.AutoScalingClient;
import software.amazon.awssdk.services.autoscaling.model.AutoScalingException;
import
 software.amazon.awssdk.services.autoscaling.model.CreateAutoScalingGroupRequest;
import
 software.amazon.awssdk.services.autoscaling.model.DescribeAutoScalingGroupsRequest;
import
 software.amazon.awssdk.services.autoscaling.model.DescribeAutoScalingGroupsResponse;
import
 software.amazon.awssdk.services.autoscaling.model.LaunchTemplateSpecification;
import software.amazon.awssdk.services.autoscaling.waiters.AutoScalingWaiter;

/**
 * Before running this SDK for Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateAutoScalingGroup {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <groupName> <launchTemplateName> <serviceLinkedRoleARN>
 <vpcZoneId>

                Where:
                    groupName - The name of the Auto Scaling group.
                    launchTemplateName - The name of the launch template.\s
                    vpcZoneId - A subnet Id for a virtual private cloud (VPC)
 where instances in the Auto Scaling group can be created.
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String groupName = args[0];
        String launchTemplateName = args[1];
        String vpcZoneId = args[2];
        AutoScalingClient autoScalingClient = AutoScalingClient.builder()
                .region(Region.US_EAST_1)
                .build();

        createAutoScalingGroup(autoScalingClient, groupName, launchTemplateName,
vpcZoneId);
        autoScalingClient.close();
    }

    public static void createAutoScalingGroup(AutoScalingClient
autoScalingClient,
            String groupName,
            String launchTemplateName,
            String vpcZoneId) {

        try {
            AutoScalingWaiter waiter = autoScalingClient.waiter();
            LaunchTemplateSpecification templateSpecification =
LaunchTemplateSpecification.builder()
                    .launchTemplateName(launchTemplateName)
                    .build();

            CreateAutoScalingGroupRequest request =
CreateAutoScalingGroupRequest.builder()
                    .autoScalingGroupName(groupName)
                    .availabilityZones("us-east-1a")
                    .launchTemplate(templateSpecification)
                    .maxSize(1)
                    .minSize(1)
                    .vpcZoneIdentifier(vpcZoneId)
                    .build();

            autoScalingClient.createAutoScalingGroup(request);
            DescribeAutoScalingGroupsRequest groupsRequest =
DescribeAutoScalingGroupsRequest.builder()
                    .autoScalingGroupNames(groupName)
                    .build();

            WaiterResponse<DescribeAutoScalingGroupsResponse> waiterResponse =
waiter
                    .waitUntilGroupExists(groupsRequest);
```

```
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("Auto Scaling Group created");

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createAutoScalingGroup(
    groupName: String,
    launchTemplateNameVal: String,
    serviceLinkedRoleARNVal: String,
    vpcZoneIdVal: String,
) {
    val templateSpecification =
        LaunchTemplateSpecification {
            launchTemplateName = launchTemplateNameVal
        }

    val request =
        CreateAutoScalingGroupRequest {
            autoScalingGroupName = groupName
            availabilityZones = listOf("us-east-1a")
            launchTemplate = templateSpecification
            maxSize = 1
            minSize = 1
            vpcZoneIdentifier = vpcZoneIdVal
```

```
            serviceLinkedRoleArn = serviceLinkedRoleARNVal
        }

    // This object is required for the waiter call.
    val groupsRequestWaiter =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.createAutoScalingGroup(request)
        autoScalingClient.waitUntilGroupExists(groupsRequestWaiter)
        println("$groupName was created!")
    }
}
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function createAutoScalingGroup(
        $autoScalingGroupName,
        $availabilityZones,
        $minSize,
        $maxSize,
        $launchTemplateId
    ) {
        return $this->autoScalingClient->createAutoScalingGroup([
            'AutoScalingGroupName' => $autoScalingGroupName,
            'AvailabilityZones' => $availabilityZones,
            'MinSize' => $minSize,
            'MaxSize' => $maxSize,
            'LaunchTemplate' => [
```

```
                'LaunchTemplateId' => $launchTemplateId,
            ],
        ]);
    }
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example creates an Auto Scaling group with the specified name and attributes. The default desired capacity is the minimum size. Therefore, this Auto Scaling group launches two instances, one in each of the specified two Availability Zones.

```
New-ASAutoScalingGroup -AutoScalingGroupName my-asg -LaunchConfigurationName my-
lc -MinSize 2 -MaxSize 6 -AvailabilityZone @("us-west-2a", "us-west-2b")
```

- For API details, see CreateAutoScalingGroup in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client
```

```
    def create_group(
        self,
        group_name: str,
        group_zones: List[str],
        launch_template_name: str,
        min_size: int,
        max_size: int,
    ) -> None:
        """
        Creates an Auto Scaling group.

        :param group_name: The name to give to the group.
        :param group_zones: The Availability Zones in which instances can be
created.
        :param launch_template_name: The name of an existing Amazon EC2 launch
template.
                                     The launch template specifies the
configuration of
                                     instances that are created by auto scaling
activities.
        :param min_size: The minimum number of active instances in the group.
        :param max_size: The maximum number of active instances in the group.
        :return: None
        :raises ClientError: If there is an error creating the Auto Scaling
group.
        """
        try:
            self.autoscaling_client.create_auto_scaling_group(
                AutoScalingGroupName=group_name,
                AvailabilityZones=group_zones,
                LaunchTemplate={
                    "LaunchTemplateName": launch_template_name,
                    "Version": "$Default",
                },
                MinSize=min_size,
                MaxSize=max_size,
            )

            # Wait for the group to exist.
            waiter = self.autoscaling_client.get_waiter("group_exists")
            waiter.wait(AutoScalingGroupNames=[group_name])

            logger.info(f"Successfully created Auto Scaling group {group_name}.")
```

```
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(f"Failed to create Auto Scaling group {group_name}.")
            if error_code == "AlreadyExistsFault":
                logger.error(
                    f"An Auto Scaling group with the name '{group_name}' already
 exists. "
                    "Please use a different name or update the existing group.",
                )
            elif error_code == "LimitExceededFault":
                logger.error(
                    "The request failed because you have reached the limit "
                    "on the number of Auto Scaling groups or launch
 configurations. "
                    "Consider deleting unused resources or request a limit
 increase. "
                    "\nSee Auto Scaling Service Quota documentation here:"
                    "\n\thttps://docs.aws.amazon.com/autoscaling/ec2/userguide/
ec2-auto-scaling-quotas.html"
                )
            logger.error(f"Full error:\n\t{err}")
            raise
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn create_group(client: &Client, name: &str, id: &str) -> Result<(), Error>
 {
    client
        .create_auto_scaling_group()
```

```
        .auto_scaling_group_name(name)
        .instance_id(id)
        .min_size(1)
        .max_size(5)
        .send()
        .await?;

    println!("Created AutoScaling group");

    Ok(())
}
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateLaunchConfiguration` with an AWS SDK or CLI

The following code examples show how to use `CreateLaunchConfiguration`.

CLI

AWS CLI

Example 1: To create a launch configuration

This example creates a simple launch configuration.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large
```

This command produces no output.

For more information, see Creating a launch configuration in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To create a launch configuration with a security group, key pair, and bootrapping script

This example creates a launch configuration with a security group, a key pair, and a bootrapping script contained in the user data.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large \
    --security-groups sg-eb2af88example \
    --key-name my-key-pair \
    --user-data file://myuserdata.txt
```

This command produces no output.

For more information, see Creating a launch configuration in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To create a launch configuration with an IAM role

This example creates a launch configuration with the instance profile name of an IAM role.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large \
    --iam-instance-profile my-autoscaling-role
```

This command produces no output.

For more information, see IAM role for applications that run on Amazon EC2 instances in the *Amazon EC2 Auto Scaling User Guide*.

Example 4: To create a launch configuration with detailed monitoring enabled

This example creates a launch configuration with EC2 detailed monitoring enabled, which sends EC2 metrics to CloudWatch in 1-minute periods.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
```

```
    --instance-type m5.large \
    --instance-monitoring Enabled=true
```

This command produces no output.

For more information, see Configuring monitoring for Auto Scaling instances in the *Amazon EC2 Auto Scaling User Guide*.

Example 5: To create a launch configuration that launches Spot Instances

This example creates a launch configuration that uses Spot Instances as the only purchase option.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large \
    --spot-price "0.50"
```

This command produces no output.

For more information, see Requesting Spot Instances in the *Amazon EC2 Auto Scaling User Guide*.

Example 6: To create a launch configuration using an EC2 instance

This example creates a launch configuration based on the attributes of an existing instance. It overrides the placement tenancy and whether a public IP address is set by including the `--placement-tenancy` and `--no-associate-public-ip-address` options.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc-from-instance \
    --instance-id i-0123a456700123456 \
    --instance-type m5.large \
    --no-associate-public-ip-address \
    --placement-tenancy dedicated
```

This command produces no output.

For more information, see Creating a launch configuration using an EC2 instance in the *Amazon EC2 Auto Scaling User Guide*.

Example 7: To create a launch configuration with a block device mapping for an Amazon EBS volume

This example creates a launch configuration with a block device mapping for an Amazon EBS gp3 volume with the device name `/dev/sdh` and a volume size of 20.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large \
    --block-device-mappings '[{"DeviceName":"/dev/sdh","Ebs":
{"VolumeSize":20,"VolumeType":"gp3"}}]'
```

This command produces no output.

For more information, see EBS in the *Amazon EC2 Auto Scaling API Reference*.

For information about the syntax for quoting JSON-formatted parameter values, see Using quotation marks with strings in the AWS CLI in the *AWS Command Line Interface User Guide*.

Example 8: To create a launch configuration with a block device mapping for an instance store volume

This example creates a launch configuration with `ephemeral1` as an instance store volume with the device name `/dev/sdc`.

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large \
    --block-device-mappings '[{"DeviceName":"/dev/
sdc","VirtualName":"ephemeral1"}]'
```

This command produces no output.

For more information, see BlockDeviceMapping in the *Amazon EC2 Auto Scaling API Reference*.

For information about the syntax for quoting JSON-formatted parameter values, see Using quotation marks with strings in the AWS CLI in the *AWS Command Line Interface User Guide*.

Example 9: To create a launch configuration and suppress a block device from attaching at launch time

This example creates a launch configuration that suppresses a block device specified by the block device mapping of the AMI (for example, `/dev/sdf`).

```
aws autoscaling create-launch-configuration \
    --launch-configuration-name my-lc \
    --image-id ami-04d5cc9b88example \
    --instance-type m5.large \
    --block-device-mappings '[{"DeviceName":"/dev/sdf","NoDevice":""}]'
```

This command produces no output.

For more information, see BlockDeviceMapping in the *Amazon EC2 Auto Scaling API Reference*.

For information about the syntax for quoting JSON-formatted parameter values, see Using quotation marks with strings in the AWS CLI in the *AWS Command Line Interface User Guide*.

- For API details, see CreateLaunchConfiguration in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example creates a launch configuration named 'my-lc'. The EC2 instances launched by Auto Scaling groups that use this launch configuration use specified instance type, AMI, security group, and IAM role.

```
New-ASLaunchConfiguration -LaunchConfigurationName my-lc -InstanceType
 "m3.medium" -ImageId "ami-12345678" -SecurityGroup "sg-12345678" -
IamInstanceProfile "myIamRole"
```

- For API details, see CreateLaunchConfiguration in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateOrUpdateTags` with an AWS SDK or CLI

The following code examples show how to use `CreateOrUpdateTags`.

CLI

AWS CLI

To create or update tags for an Auto Scaling group

This example adds two tags to the specified Auto Scaling group.

```
aws autoscaling create-or-update-tags \
    --tags ResourceId=my-asg,ResourceType=auto-scaling-
group,Key=Role,Value=WebServer,PropagateAtLaunch=true ResourceId=my-
asg,ResourceType=auto-scaling-
group,Key=Dept,Value=Research,PropagateAtLaunch=true
```

This command produces no output.

For more information, see Tagging Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see CreateOrUpdateTags in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example adds a single tag to the specified Auto Scaling group. The tag key is 'myTag' and the tag value is 'myTagValue'. Auto Scaling propagates this tag to the subsequent EC2 instances launched by the Auto Scaling group. The syntax used by this example requires PowerShell version 3 or later.

```
Set-ASTag -Tag @( @{ResourceType="auto-scaling-group"; ResourceId="my-asg";
 Key="myTag"; Value="myTagValue"; PropagateAtLaunch=$true} )
```

Example 2: With PowerShell version 2, you must use New-Object to create the tag for the Tag parameter.

```
$tag = New-Object Amazon.AutoScaling.Model.Tag
$tag.ResourceType = "auto-scaling-group"
```

```
$tag.ResourceId = "my-asg"
$tag.Key = "myTag"
$tag.Value = "myTagValue"
$tag.PropagateAtLaunch = $true
Set-ASTag -Tag $tag
```

- For API details, see CreateOrUpdateTags in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAutoScalingGroup` with an AWS SDK or CLI

The following code examples show how to use `DeleteAutoScalingGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Build and manage a resilient service

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Update the minimum size of an Auto Scaling group to zero, terminate all instances in the group, and delete the group.

```
    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
```

```
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for
 {instanceId}. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
 progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
```

```
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string
groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
```

```
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }
```

```
    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

            Aws::AutoScaling::Model::DeleteAutoScalingGroupRequest request;
            request.SetAutoScalingGroupName(groupName);

            Aws::AutoScaling::Model::DeleteAutoScalingGroupOutcome outcome =
                    autoScalingClient.DeleteAutoScalingGroup(request);

            if (outcome.IsSuccess()) {
                std::cout << "Auto Scaling group '" << groupName << "' was
deleted."
                          << std::endl;
            }
            else {
                std::cerr << "Error with AutoScaling::DeleteAutoScalingGroup. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To delete the specified Auto Scaling group

This example deletes the specified Auto Scaling group.

```
aws autoscaling delete-auto-scaling-group \
    --auto-scaling-group-name my-asg
```

This command produces no output.

For more information, see Deleting your Auto Scaling infrastructure in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To force delete the specified Auto Scaling group

To delete the Auto Scaling group without waiting for the instances in the group to terminate, use the `--force-delete` option.

```
aws autoscaling delete-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --force-delete
```

This command produces no output.

For more information, see Deleting your Auto Scaling infrastructure in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DeleteAutoScalingGroup in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.autoscaling.AutoScalingClient;
import software.amazon.awssdk.services.autoscaling.model.AutoScalingException;
import
 software.amazon.awssdk.services.autoscaling.model.DeleteAutoScalingGroupRequest;

/**
 * Before running this SDK for Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteAutoScalingGroup {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <groupName>

                Where:
                    groupName - The name of the Auto Scaling group.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String groupName = args[0];
        AutoScalingClient autoScalingClient = AutoScalingClient.builder()
                .region(Region.US_EAST_1)
                .build();

        deleteAutoScalingGroup(autoScalingClient, groupName);
        autoScalingClient.close();
    }

    public static void deleteAutoScalingGroup(AutoScalingClient
 autoScalingClient, String groupName) {
        try {
```

```
            DeleteAutoScalingGroupRequest deleteAutoScalingGroupRequest =
 DeleteAutoScalingGroupRequest.builder()
                    .autoScalingGroupName(groupName)
                    .forceDelete(true)
                    .build();


 autoScalingClient.deleteAutoScalingGroup(deleteAutoScalingGroupRequest);
            System.out.println("You successfully deleted " + groupName);

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteSpecificAutoScalingGroup(groupName: String) {
    val deleteAutoScalingGroupRequest =
        DeleteAutoScalingGroupRequest {
            autoScalingGroupName = groupName
            forceDelete = true
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.deleteAutoScalingGroup(deleteAutoScalingGroupRequest)
        println("You successfully deleted $groupName")
    }
}
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function deleteAutoScalingGroup($autoScalingGroupName)
    {
        return $this->autoScalingClient->deleteAutoScalingGroup([
            'AutoScalingGroupName' => $autoScalingGroupName,
            'ForceDelete' => true,
        ]);
    }
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified Auto Scaling group if it has no running instances. You are prompted for confirmation before the operation proceeds.

```
Remove-ASAutoScalingGroup -AutoScalingGroupName my-asg
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASAutoScalingGroup (DeleteAutoScalingGroup)" on
 Target "my-asg".
```

```
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASAutoScalingGroup -AutoScalingGroupName my-asg -Force
```

Example 3: This example deletes the specified Auto Scaling group and terminates any running instances that it contains.

```
Remove-ASAutoScalingGroup -AutoScalingGroupName my-asg -ForceDelete $true -Force
```

- For API details, see DeleteAutoScalingGroup in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Update the minimum size of an Auto Scaling group to zero, terminate all instances in the group, and delete the group.

```
class AutoScalingWrapper:
    """
    Encapsulates Amazon EC2 Auto Scaling and EC2 management actions.
    """

    def __init__(
        self,
        resource_prefix: str,
        inst_type: str,
        ami_param: str,
        autoscaling_client: boto3.client,
        ec2_client: boto3.client,
```

```
        ssm_client: boto3.client,
        iam_client: boto3.client,
    ):
        """
        Initializes the AutoScaler class with the necessary parameters.

        :param resource_prefix: The prefix for naming AWS resources that are
created by this class.
        :param inst_type: The type of EC2 instance to create, such as t3.micro.
        :param ami_param: The Systems Manager parameter used to look up the AMI
that is created.
        :param autoscaling_client: A Boto3 EC2 Auto Scaling client.
        :param ec2_client: A Boto3 EC2 client.
        :param ssm_client: A Boto3 Systems Manager client.
        :param iam_client: A Boto3 IAM client.
        """
        self.inst_type = inst_type
        self.ami_param = ami_param
        self.autoscaling_client = autoscaling_client
        self.ec2_client = ec2_client
        self.ssm_client = ssm_client
        self.iam_client = iam_client
        sts_client = boto3.client("sts")
        self.account_id = sts_client.get_caller_identity()["Account"]

        self.key_pair_name = f"{resource_prefix}-key-pair"
        self.launch_template_name = f"{resource_prefix}-template-"
        self.group_name = f"{resource_prefix}-group"

        # Happy path
        self.instance_policy_name = f"{resource_prefix}-pol"
        self.instance_role_name = f"{resource_prefix}-role"
        self.instance_profile_name = f"{resource_prefix}-prof"

        # Failure mode
        self.bad_creds_policy_name = f"{resource_prefix}-bc-pol"
        self.bad_creds_role_name = f"{resource_prefix}-bc-role"
        self.bad_creds_profile_name = f"{resource_prefix}-bc-prof"


    def delete_autoscaling_group(self, group_name: str) -> None:
        """
        Terminates all instances in the group, then deletes the EC2 Auto Scaling
group.
```

```
        :param group_name: The name of the group to delete.
        """
        try:
            response = self.autoscaling_client.describe_auto_scaling_groups(
                AutoScalingGroupNames=[group_name]
            )
            groups = response.get("AutoScalingGroups", [])
            if len(groups) > 0:
                self.autoscaling_client.update_auto_scaling_group(
                    AutoScalingGroupName=group_name, MinSize=0
                )
                instance_ids = [inst["InstanceId"] for inst in groups[0]
["Instances"]]
                for inst_id in instance_ids:
                    self.terminate_instance(inst_id)

                # Wait for all instances to be terminated
                if instance_ids:
                    waiter = self.ec2_client.get_waiter("instance_terminated")
                    log.info("Waiting for all instances to be terminated...")
                    waiter.wait(InstanceIds=instance_ids)
                    log.info("All instances have been terminated.")
            else:
                log.info(f"No groups found named '{group_name}'! Nothing to do.")
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to delete Auto Scaling group '{group_name}'.")
            if error_code == "ScalingActivityInProgressFault":
                log.error(
                    "Scaling activity is currently in progress. "
                    "Wait for the scaling activity to complete before attempting
 to delete the group again."
                )
            elif error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
 the group."
                )
            log.error(f"Full error:\n\t{err}")
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn delete_group(client: &Client, name: &str, force: bool) -> Result<(),
 Error> {
    client
        .delete_auto_scaling_group()
        .auto_scaling_group_name(name)
        .set_force_delete(if force { Some(true) } else { None })
        .send()
        .await?;

    println!("Deleted Auto Scaling group");

    Ok(())
}
```

- For API details, see DeleteAutoScalingGroup in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteLaunchConfiguration` with an AWS SDK or CLI

The following code examples show how to use `DeleteLaunchConfiguration`.

CLI

AWS CLI

To delete a launch configuration

This example deletes the specified launch configuration.

```
aws autoscaling delete-launch-configuration \
    --launch-configuration-name my-launch-config
```

This command produces no output.

For more information, see Deleting your Auto Scaling infrastructure in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DeleteLaunchConfiguration in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified launch configuration if it is not attached to an Auto Scaling group. You are prompted for confirmation before the operation proceeds.

```
Remove-ASLaunchConfiguration -LaunchConfigurationName my-lc
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASLaunchConfiguration (DeleteLaunchConfiguration)"
 on Target "my-lc".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASLaunchConfiguration -LaunchConfigurationName my-lc -Force
```

- For API details, see DeleteLaunchConfiguration in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteLifecycleHook` with an AWS SDK or CLI

The following code examples show how to use `DeleteLifecycleHook`.

CLI

AWS CLI

To delete a lifecycle hook

This example deletes the specified lifecycle hook.

```
aws autoscaling delete-lifecycle-hook \
    --lifecycle-hook-name my-lifecycle-hook \
    --auto-scaling-group-name my-asg
```

This command produces no output.

- For API details, see DeleteLifecycleHook in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified lifecycle hook for the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds.

```
Remove-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName
 myLifecycleHook
```

Output:

```
Confirm
```

```
Are you sure you want to perform this action?
Performing operation "Remove-ASLifecycleHook (DeleteLifecycleHook)" on Target
 "myLifecycleHook".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName
 myLifecycleHook -Force
```

- For API details, see DeleteLifecycleHook in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteNotificationConfiguration` with an AWS SDK or CLI

The following code examples show how to use `DeleteNotificationConfiguration`.

CLI

AWS CLI

To delete an Auto Scaling notification

This example deletes the specified notification from the specified Auto Scaling group.

```
aws autoscaling delete-notification-configuration \
    --auto-scaling-group-name my-asg \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-sns-topic
```

This command produces no output.

For more information, see Delete the notification configuration in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DeleteNotificationConfiguration in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified notification action. You are prompted for confirmation before the operation proceeds.

```
Remove-ASNotificationConfiguration -AutoScalingGroupName my-asg -TopicARN
 "arn:aws:sns:us-west-2:123456789012:my-topic"
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASNotificationConfiguration
 (DeleteNotificationConfiguration)" on Target
"arn:aws:sns:us-west-2:123456789012:my-topic".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASNotificationConfiguration -AutoScalingGroupName my-asg -TopicARN
 "arn:aws:sns:us-west-2:123456789012:my-topic" -Force
```

- For API details, see DeleteNotificationConfiguration in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeletePolicy` with an AWS SDK or CLI

The following code examples show how to use `DeletePolicy`.

CLI

AWS CLI

To delete a scaling policy

This example deletes the specified scaling policy.

```
aws autoscaling delete-policy \
    --auto-scaling-group-name my-asg \
    --policy-name alb1000-target-tracking-scaling-policy
```

This command produces no output.

- For API details, see DeletePolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified policy for the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds.

```
Remove-ASPolicy -AutoScalingGroupName my-asg -PolicyName myScaleInPolicy
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASPolicy (DeletePolicy)" on Target
 "myScaleInPolicy".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASPolicy -AutoScalingGroupName my-asg -PolicyName myScaleInPolicy -Force
```

- For API details, see DeletePolicy in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteScheduledAction` with an AWS SDK or CLI

The following code examples show how to use `DeleteScheduledAction`.

CLI

AWS CLI

To delete a scheduled action from an Auto Scaling group

This example deletes the specified scheduled action from the specified Auto Scaling group.

```
aws autoscaling delete-scheduled-action \
    --auto-scaling-group-name my-asg \
    --scheduled-action-name my-scheduled-action
```

This command produces no output.

- For API details, see DeleteScheduledAction in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified scheduled action for the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds.

```
Remove-ASScheduledAction -AutoScalingGroupName my-asg -ScheduledAction
 "myScheduledAction"
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove-ASScheduledAction (DeleteScheduledAction)" on Target
 "myScheduledAction".
```

```
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASScheduledAction -AutoScalingGroupName my-asg -ScheduledAction
 "myScheduledAction" -Force
```

- For API details, see DeleteScheduledAction in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteTags` with an AWS SDK or CLI

The following code examples show how to use `DeleteTags`.

CLI

AWS CLI

To delete a tag from an Auto Scaling group

This example deletes the specified tag from the specified Auto Scaling group.

```
aws autoscaling delete-tags \
    --tags ResourceId=my-asg,ResourceType=auto-scaling-
group,Key=Dept,Value=Research
```

This command produces no output.

For more information, see Tagging Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DeleteTags in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example removes the specified tag from the specified Auto Scaling group. You are prompted for confirmation before the operation proceeds. The syntax used by this example requires PowerShell version 3 or later.

```
Remove-ASTag -Tag @( @{ResourceType="auto-scaling-group"; ResourceId="my-asg";
 Key="myTag" } )
```

Output:

```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove-ASTag (DeleteTags)" on target
 "Amazon.AutoScaling.Model.Tag".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is
 "Y"):
```

Example 2: If you specify the Force parameter, you are not prompted for confirmation before the operation proceeds.

```
Remove-ASTag -Tag @( @{ResourceType="auto-scaling-group"; ResourceId="my-asg";
 Key="myTag" } ) -Force
```

Example 3: With Powershell version 2, you must use New-Object to create the tag for the Tag parameter.

```
$tag = New-Object Amazon.AutoScaling.Model.Tag
$tag.ResourceType = "auto-scaling-group"
$tag.ResourceId = "my-asg"
$tag.Key = "myTag"
Remove-ASTag -Tag $tag -Force
```

- For API details, see DeleteTags in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAccountLimits` with an AWS SDK or CLI

The following code examples show how to use `DescribeAccountLimits`.

CLI

AWS CLI

To describe your Amazon EC2 Auto Scaling account limits

This example describes the Amazon EC2 Auto Scaling limits for your AWS account.

```
aws autoscaling describe-account-limits
```

Output:

```
{
    "NumberOfLaunchConfigurations": 5,
    "MaxNumberOfLaunchConfigurations": 100,
    "NumberOfAutoScalingGroups": 3,
    "MaxNumberOfAutoScalingGroups": 20
}
```

For more information, see Amazon EC2 Auto Scaling service quotas in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeAccountLimits in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the Auto Scaling resource limits for your AWS account.

```
Get-ASAccountLimit
```

Output:

```
MaxNumberOfAutoScalingGroups    : 20
MaxNumberOfLaunchConfigurations : 100
```

- For API details, see DescribeAccountLimits in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAdjustmentTypes` with an AWS SDK or CLI

The following code examples show how to use `DescribeAdjustmentTypes`.

CLI

AWS CLI

To describe the available scaling adjustment types

This example describes the available adjustment types.

```
aws autoscaling describe-adjustment-types
```

Output:

```
{
    "AdjustmentTypes": [
        {
            "AdjustmentType": "ChangeInCapacity"
        },
        {
            "AdjustmentType": "ExactCapacity"
        },
        {
            "AdjustmentType": "PercentChangeInCapacity"
        }
    ]
}
```

For more information, see Scaling adjustment types in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeAdjustmentTypes in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the adjustment types that are supported by Auto Scaling.

```
Get-ASAdjustmentType
```

Output:

```
Type
----
ChangeInCapacity
ExactCapacity
PercentChangeInCapacity
```

- For API details, see DescribeAdjustmentTypes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAutoScalingGroups` with an AWS SDK or CLI

The following code examples show how to use `DescribeAutoScalingGroups`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Build and manage a resilient service

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
```

```
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

    Aws::AutoScaling::Model::DescribeAutoScalingGroupsRequest request;
    Aws::Vector<Aws::String> groupNames;
    groupNames.push_back(groupName);
    request.SetAutoScalingGroupNames(groupNames);

    Aws::AutoScaling::Model::DescribeAutoScalingGroupsOutcome outcome =
            client.DescribeAutoScalingGroups(request);

    if (outcome.IsSuccess()) {
        autoScalingGroup = outcome.GetResult().GetAutoScalingGroups();
    }
    else {
        std::cerr << "Error with AutoScaling::DescribeAutoScalingGroups. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To describe the specified Auto Scaling group

This example describes the specified Auto Scaling group.

```
aws autoscaling describe-auto-scaling-groups \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "my-asg",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:930d940e-891e-4781-
a11a-7b0acd480f03:autoScalingGroupName/my-asg",
            "LaunchTemplate": {
                "LaunchTemplateName": "my-launch-template",
                "Version": "1",
                "LaunchTemplateId": "lt-1234567890abcde12"
            },
            "MinSize": 0,
            "MaxSize": 1,
            "DesiredCapacity": 1,
            "DefaultCooldown": 300,
            "AvailabilityZones": [
                "us-west-2a",
                "us-west-2b",
                "us-west-2c"
            ],
            "LoadBalancerNames": [],
            "TargetGroupARNs": [],
            "HealthCheckType": "EC2",
            "HealthCheckGracePeriod": 0,
```

```
            "Instances": [
                {
                    "InstanceId": "i-06905f55584de02da",
                    "InstanceType": "t2.micro",
                    "AvailabilityZone": "us-west-2a",
                    "HealthStatus": "Healthy",
                    "LifecycleState": "InService",
                    "ProtectedFromScaleIn": false,
                    "LaunchTemplate": {
                        "LaunchTemplateName": "my-launch-template",
                        "Version": "1",
                        "LaunchTemplateId": "lt-1234567890abcde12"
                    }
                }
            ],
            "CreatedTime": "2023-10-28T02:39:22.152Z",
            "SuspendedProcesses": [],
            "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-
c934b782",
            "EnabledMetrics": [],
            "Tags": [],
            "TerminationPolicies": [
                "Default"
            ],
            "NewInstancesProtectedFromScaleIn": false,
            "ServiceLinkedRoleARN":"arn",
            "TrafficSources": []
        }
    ]
}
```

Example 2: To describe the first 100 specified Auto Scaling group

This example describes the specified Auto Scaling groups. It allows you to specify up to 100 group names.

```
aws autoscaling describe-auto-scaling-groups \
    --max-items 100 \
    --auto-scaling-group-name "group1" "group2" "group3" "group4"
```

See example 1 for sample output.

Example 3: To describe an Auto Scaling group in the specified region

This example describes the Auto Scaling groups in the specified region, up to a maximum of 75 groups.

```
aws autoscaling describe-auto-scaling-groups \
    --max-items 75 \
    --region us-east-1
```

See example 1 for sample output.

Example 4: To describe the specified number of Auto Scaling group

To return a specific number of Auto Scaling groups, use the `--max-items` option.

```
aws autoscaling describe-auto-scaling-groups \
    --max-items 1
```

See example 1 for sample output.

If the output includes a `NextToken` field, there are more groups. To get the additional groups, use the value of this field with the `--starting-token` option in a subsequent call as follows.

```
aws autoscaling describe-auto-scaling-groups \
    --starting-token Z3M3LMPEXAMPLE
```

See example 1 for sample output.

Example 5: To describe Auto Scaling groups that use launch configurations

This example uses the `--query` option to describe Auto Scaling groups that use launch configurations.

```
aws autoscaling describe-auto-scaling-groups \
    --query 'AutoScalingGroups[?LaunchConfigurationName!=`null`]'
```

Output:

```
[
    {
        "AutoScalingGroupName": "my-asg",
```

```
        "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:930d940e-891e-4781-
a11a-7b0acd480f03:autoScalingGroupName/my-asg",
        "LaunchConfigurationName": "my-lc",
        "MinSize": 0,
        "MaxSize": 1,
        "DesiredCapacity": 1,
        "DefaultCooldown": 300,
        "AvailabilityZones": [
            "us-west-2a",
            "us-west-2b",
            "us-west-2c"
        ],
        "LoadBalancerNames": [],
        "TargetGroupARNs": [],
        "HealthCheckType": "EC2",
        "HealthCheckGracePeriod": 0,
        "Instances": [
            {
                "InstanceId": "i-088c57934a6449037",
                "InstanceType": "t2.micro",
                "AvailabilityZone": "us-west-2c",
                "HealthStatus": "Healthy",
                "LifecycleState": "InService",
                "LaunchConfigurationName": "my-lc",
                "ProtectedFromScaleIn": false
            }
        ],
        "CreatedTime": "2023-10-28T02:39:22.152Z",
        "SuspendedProcesses": [],
        "VPCZoneIdentifier": "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782",
        "EnabledMetrics": [],
        "Tags": [],
        "TerminationPolicies": [
            "Default"
        ],
        "NewInstancesProtectedFromScaleIn": false,
        "ServiceLinkedRoleARN":"arn",
        "TrafficSources": []
    }
]
```

For more information, see Filter AWS CLI output in the *AWS Command Line Interface User Guide*.

- For API details, see DescribeAutoScalingGroups in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.autoscaling.AutoScalingClient;
import software.amazon.awssdk.services.autoscaling.model.AutoScalingException;
import software.amazon.awssdk.services.autoscaling.model.AutoScalingGroup;
import
 software.amazon.awssdk.services.autoscaling.model.DescribeAutoScalingGroupsResponse;
import
 software.amazon.awssdk.services.autoscaling.model.DescribeAutoScalingGroupsRequest;
import software.amazon.awssdk.services.autoscaling.model.Instance;
import java.util.List;

/**
 * Before running this SDK for Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeAutoScalingInstances {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <groupName>
```

```
                Where:
                    groupName - The name of the Auto Scaling group.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String groupName = args[0];
        AutoScalingClient autoScalingClient = AutoScalingClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String instanceId = getAutoScaling(autoScalingClient, groupName);
        System.out.println(instanceId);
        autoScalingClient.close();
    }

    public static String getAutoScaling(AutoScalingClient autoScalingClient,
String groupName) {
        try {
            String instanceId = "";
            DescribeAutoScalingGroupsRequest scalingGroupsRequest =
DescribeAutoScalingGroupsRequest.builder()
                    .autoScalingGroupNames(groupName)
                    .build();

            DescribeAutoScalingGroupsResponse response = autoScalingClient
                    .describeAutoScalingGroups(scalingGroupsRequest);
            List<AutoScalingGroup> groups = response.autoScalingGroups();
            for (AutoScalingGroup group : groups) {
                System.out.println("The group name is " +
group.autoScalingGroupName());
                System.out.println("The group ARN is " +
group.autoScalingGroupARN());

                List<Instance> instances = group.instances();
                for (Instance instance : instances) {
                    instanceId = instance.instanceId();
                }
            }
            return instanceId;
        } catch (AutoScalingException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAutoScalingGroups(groupName: String) {
    val scalingGroupsRequest =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeAutoScalingGroups(scalingGroupsRequest)
        response.autoScalingGroups?.forEach { group ->
            println("The group name is ${group.autoScalingGroupName}")
            println("The group ARN is ${group.autoScalingGroupArn}")
            group.instances?.forEach { instance ->
                println("The instance id is ${instance.instanceId}")
                println("The lifecycle state is " + instance.lifecycleState)
            }
        }
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function describeAutoScalingGroups($autoScalingGroupNames)
    {
        return $this->autoScalingClient->describeAutoScalingGroups([
            'AutoScalingGroupNames' => $autoScalingGroupNames
        ]);
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the names of your Auto Scaling groups.

```
Get-ASAutoScalingGroup | format-table -property AutoScalingGroupName
```

Output:

```
AutoScalingGroupName
--------------------
my-asg-1
my-asg-2
my-asg-3
my-asg-4
my-asg-5
my-asg-6
```

Example 2: This example describes the specified Auto Scaling group.

```
Get-ASAutoScalingGroup -AutoScalingGroupName my-asg-1
```

Output:

```
AutoScalingGroupARN     : arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:930d940e-891e-4781-a11a-7b0acd480
                          f03:autoScalingGroupName/my-asg-1
AutoScalingGroupName    : my-asg-1
AvailabilityZones       : {us-west-2b, us-west-2a}
CreatedTime             : 3/1/2015 9:05:31 AM
DefaultCooldown         : 300
DesiredCapacity         : 2
EnabledMetrics          : {}
HealthCheckGracePeriod  : 300
HealthCheckType         : EC2
Instances               : {my-lc}
LaunchConfigurationName : my-lc
LoadBalancerNames       : {}
MaxSize                 : 0
MinSize                 : 0
PlacementGroup          :
Status                  :
SuspendedProcesses      : {}
Tags                    : {}
TerminationPolicies     : {Default}
VPCZoneIdentifier       : subnet-e4f33493,subnet-5264e837
```

Example 3: This example describes the specified two Auto Scaling groups.

```
Get-ASAutoScalingGroup -AutoScalingGroupName @("my-asg-1", "my-asg-2")
```

Example 4: This example describes the Auto Scaling instances for the specified Auto Scaling group.

```
(Get-ASAutoScalingGroup -AutoScalingGroupName my-asg-1).Instances
```

Example 5: This example describes all your Auto Scaling groups.

```
Get-ASAutoScalingGroup
```

Example 6: This example describes all your Auto Scaling groups, in batches of 10.

```
$nextToken = $null
do {
  Get-ASAutoScalingGroup -NextToken $nextToken -MaxRecord 10
  $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

Example 7: This example describes LaunchTemplate for the specified Auto Scaling group. This example assumes that the "Instance purchase options" is set to "Adhere to launch template". In case this option is set to "Combine purchase options and instance types", LaunchTemplate could be accessed using "MixedInstancesPolicy.LaunchTemplate" property.

```
(Get-ASAutoScalingGroup -AutoScalingGroupName my-ag-1).LaunchTemplate
```

Output:

```
LaunchTemplateId     LaunchTemplateName   Version
----------------     ------------------   -------
lt-06095fd619cb40371 test-launch-template $Default
```

- For API details, see DescribeAutoScalingGroups in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
```

```
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def describe_group(self, group_name: str) -> Optional[Dict[str, Any]]:
        """
        Gets information about an Auto Scaling group.

        :param group_name: The name of the group to look up.
        :return: A dictionary with information about the group if found,
otherwise None.
        :raises ClientError: If there is an error describing the Auto Scaling
group.
        """
        try:
            paginator = self.autoscaling_client.get_paginator(
                "describe_auto_scaling_groups"
            )
            response_iterator =
paginator.paginate(AutoScalingGroupNames=[group_name])
            groups = []
            for response in response_iterator:
                groups.extend(response.get("AutoScalingGroups", []))

            logger.info(
                f"Successfully retrieved information for Auto Scaling group
{group_name}."
            )

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(f"Failed to describe Auto Scaling group {group_name}.")
            if error_code == "ResourceContentionFault":
                logger.error(
                    "There is a conflict with another operation that is modifying
the "
                    f"Auto Scaling group '{group_name}' Please try again later."
                )
            logger.error(f"Full error:\n\t{err}")
            raise
        else:
            return groups[0] if len(groups) > 0 else None
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn list_groups(client: &Client) -> Result<(), Error> {
    let resp = client.describe_auto_scaling_groups().send().await?;

    println!("Groups:");

    let groups = resp.auto_scaling_groups();

    for group in groups {
        println!(
            "Name:  {}",
            group.auto_scaling_group_name().unwrap_or("Unknown")
        );
        println!(
            "Arn:   {}",
            group.auto_scaling_group_arn().unwrap_or("unknown"),
        );
        println!("Zones: {:?}", group.availability_zones(),);
        println!();
    }

    println!("Found {} group(s)", groups.len());

    Ok(())
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAutoScalingInstances` with an AWS SDK or CLI

The following code examples show how to use `DescribeAutoScalingInstances`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        groups!.ForEach(group =>
        {
```

```
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
_amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);
```

```
        Aws::AutoScaling::Model::DescribeAutoScalingInstancesRequest request;
        request.SetInstanceIds(instanceIDs);

        Aws::AutoScaling::Model::DescribeAutoScalingInstancesOutcome outcome =
                client.DescribeAutoScalingInstances(request);

        if (outcome.IsSuccess()) {
            const
 Aws::Vector<Aws::AutoScaling::Model::AutoScalingInstanceDetails>
 &instancesDetails =
                   outcome.GetResult().GetAutoScalingInstances();

        }
        else {
            std::cerr << "Error with AutoScaling::DescribeAutoScalingInstances. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To describe one or more instances

This example describes the specified instance.

```
aws autoscaling describe-auto-scaling-instances \
    --instance-ids i-06905f55584de02da
```

Output:

```
{
    "AutoScalingInstances": [
        {
            "InstanceId": "i-06905f55584de02da",
            "InstanceType": "t2.micro",
            "AutoScalingGroupName": "my-asg",
```

```
            "AvailabilityZone": "us-west-2b",
            "LifecycleState": "InService",
            "HealthStatus": "HEALTHY",
            "ProtectedFromScaleIn": false,
            "LaunchTemplate": {
                "LaunchTemplateId": "lt-1234567890abcde12",
                "LaunchTemplateName": "my-launch-template",
                "Version": "1"
            }
        }
    ]
}
```

Example 2: To describe one or more instances

This example uses the `--max-items` option to specify how many instances to return with this call.

```
aws autoscaling describe-auto-scaling-instances \
    --max-items 1
```

If the output includes a `NextToken` field, there are more instances. To get the additional instances, use the value of this field with the `--starting-token` option in a subsequent call as follows.

```
aws autoscaling describe-auto-scaling-instances \
    --starting-token Z3M3LMPEXAMPLE
```

See example 1 for sample output.

Example 3: To describe instances that use launch configurations

This example uses the `--query` option to describe instances that use launch configurations.

```
aws autoscaling describe-auto-scaling-instances \
    --query 'AutoScalingInstances[?LaunchConfigurationName!=`null`]'
```

Output:

```
[
```

```
    {
        "InstanceId": "i-088c57934a6449037",
        "InstanceType": "t2.micro",
        "AutoScalingGroupName": "my-asg",
        "AvailabilityZone": "us-west-2c",
        "LifecycleState": "InService",
        "HealthStatus": "HEALTHY",
        "LaunchConfigurationName": "my-lc",
        "ProtectedFromScaleIn": false
    }
]
```

For more information, see Filter AWS CLI output in the *AWS Command Line Interface User Guide*.

- For API details, see DescribeAutoScalingInstances in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void describeAutoScalingInstance(AutoScalingClient
autoScalingClient, String id) {
       try {
           DescribeAutoScalingInstancesRequest
describeAutoScalingInstancesRequest = DescribeAutoScalingInstancesRequest
                   .builder()
                   .instanceIds(id)
                   .build();

           DescribeAutoScalingInstancesResponse response = autoScalingClient

.describeAutoScalingInstances(describeAutoScalingInstancesRequest);
           List<AutoScalingInstanceDetails> instances =
response.autoScalingInstances();
           for (AutoScalingInstanceDetails instance : instances) {
```

```
                System.out.println("The instance lifecycle state is: " +
 instance.lifecycleState());
            }

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun describeAutoScalingInstance(id: String) {
    val describeAutoScalingInstancesRequest =
        DescribeAutoScalingInstancesRequest {
            instanceIds = listOf(id)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response =
 autoScalingClient.describeAutoScalingInstances(describeAutoScalingInstancesRequest)
        response.autoScalingInstances?.forEach { group ->
            println("The instance lifecycle state is: ${group.lifecycleState}")
        }
    }
}
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function describeAutoScalingInstances($instanceIds)
    {
        return $this->autoScalingClient->describeAutoScalingInstances([
            'InstanceIds' => $instanceIds
        ]);
    }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the IDs of your Auto Scaling instances.

```
Get-ASAutoScalingInstance | format-table -property InstanceId
```

Output:

```
InstanceId
----------
i-12345678
i-87654321
i-abcd1234
```

Example 2: This example describes the specified Auto Scaling instance.

```
Get-ASAutoScalingInstance -InstanceId i-12345678
```

Output:

```
AutoScalingGroupName    : my-asg
AvailabilityZone        : us-west-2b
HealthStatus            : HEALTHY
InstanceId              : i-12345678
LaunchConfigurationName : my-lc
LifecycleState          : InService
```

Example 3: This example describes the specified two Auto Scaling instances.

```
Get-ASAutoScalingInstance -InstanceId @("i-12345678", "i-87654321")
```

Example 4: This example describes the Auto Scaling instances for the specified Auto Scaling group.

```
(Get-ASAutoScalingGroup -AutoScalingGroupName my-asg).Instances | Get-
ASAutoScalingInstance
```

Example 5: This example describes all your Auto Scaling instances.

```
Get-ASAutoScalingInstance
```

Example 6: This example describes all your Auto Scaling instances, in batches of 10.

```
$nextToken = $null
do {
  Get-ASAutoScalingInstance -NextToken $nextToken -MaxRecord 10
  $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

- For API details, see DescribeAutoScalingInstances in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def describe_instances(self, instance_ids: List[str]) -> List[Dict[str,
 Any]]:
        """
        Gets information about instances.

        :param instance_ids: A list of instance IDs to look up.
        :return: A list of dictionaries with information about each instance,
                 or an empty list if none are found.
        :raises ClientError: If there is an error describing the instances.
        """
        try:
            paginator = self.autoscaling_client.get_paginator(
                "describe_auto_scaling_instances"
            )
            response_iterator = paginator.paginate(InstanceIds=instance_ids)

            instances = []
            for response in response_iterator:
                instances.extend(response.get("AutoScalingInstances", []))

            logger.info(f"Successfully described instances: {instance_ids}")
```

```
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(
                f"Couldn't describe instances {instance_ids}. Error code:
 {error_code}, Message: {err.response['Error']['Message']}"
            )
            raise
        else:
            return instances
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    pub async fn list_instances(&self) -> Result<Vec<String>, ScenarioError> {
        // The direct way to list instances is by using
 DescribeAutoScalingGroup's instances property. However, this returns a
 Vec<Instance>, as opposed to a Vec<AutoScalingInstanceDetails>.
        // Ok(self.get_group().await?.instances.unwrap_or_default().map(|
i| i.instance_id.clone().unwrap_or_default()).filter(|id| !
id.is_empty()).collect())

        // Alternatively, and for the sake of example,
 DescribeAutoScalingInstances returns a list that can be filtered by the client.
        self.autoscaling
            .describe_auto_scaling_instances()
            .into_paginator()
            .items()
            .send()
            .try_collect()
```

```
            .await
            .map(|items| {
                items
                    .into_iter()
                    .filter(|i| {
                        i.auto_scaling_group_name.as_deref()
                            == Some(self.auto_scaling_group_name.as_str())
                    })
                    .map(|i| i.instance_id.unwrap_or_default())
                    .filter(|id| !id.is_empty())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Failed to get list of auto scaling
instances", &err))
   }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAutoScalingNotificationTypes` with an AWS SDK or CLI

The following code examples show how to use `DescribeAutoScalingNotificationTypes`.

CLI

AWS CLI

To describe the available notification types

This example describes the available notification types.

```
aws autoscaling describe-auto-scaling-notification-types
```

Output:

```
{
    "AutoScalingNotificationTypes": [
```

```
        "autoscaling:EC2_INSTANCE_LAUNCH",
        "autoscaling:EC2_INSTANCE_LAUNCH_ERROR",
        "autoscaling:EC2_INSTANCE_TERMINATE",
        "autoscaling:EC2_INSTANCE_TERMINATE_ERROR",
        "autoscaling:TEST_NOTIFICATION"
    ]
}
```

For more information, see Getting Amazon SNS notifications when your Auto Scaling group scales in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeAutoScalingNotificationTypes in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the notification types that are supported by Auto Scaling.

```
Get-ASAutoScalingNotificationType
```

Output:

```
autoscaling:EC2_INSTANCE_LAUNCH
autoscaling:EC2_INSTANCE_LAUNCH_ERROR
autoscaling:EC2_INSTANCE_TERMINATE
autoscaling:EC2_INSTANCE_TERMINATE_ERROR
autoscaling:TEST_NOTIFICATION
```

- For API details, see DescribeAutoScalingNotificationTypes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeLaunchConfigurations` with an AWS SDK or CLI

The following code examples show how to use `DescribeLaunchConfigurations`.

CLI

AWS CLI

Example 1: To describe the specified launch configuration

This example describes the specified launch configuration.

```
aws autoscaling describe-launch-configurations \
    --launch-configuration-names my-launch-config
```

Output:

```
{
    "LaunchConfigurations": [
        {
            "LaunchConfigurationName": "my-launch-config",
            "LaunchConfigurationARN": "arn:aws:autoscaling:us-
west-2:123456789012:launchConfiguration:98d3b196-4cf9-4e88-8ca1-8547c24ced8b:launchConfig
my-launch-config",
            "ImageId": "ami-0528a5175983e7f28",
            "KeyName": "my-key-pair-uswest2",
            "SecurityGroups": [
                "sg-05eaec502fcdadc2e"
            ],
            "ClassicLinkVPCSecurityGroups": [],
            "UserData": "",
            "InstanceType": "t2.micro",
            "KernelId": "",
            "RamdiskId": "",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/xvda",
                    "Ebs": {
                        "SnapshotId": "snap-06c1606ba5ca274b1",
                        "VolumeSize": 8,
                        "VolumeType": "gp2",
                        "DeleteOnTermination": true,
                        "Encrypted": false
                    }
                }
            ],
            "InstanceMonitoring": {
```

```
                "Enabled": true
            },
            "CreatedTime": "2020-10-28T02:39:22.321Z",
            "EbsOptimized": false,
            "AssociatePublicIpAddress": true,
            "MetadataOptions": {
                "HttpTokens": "required",
                "HttpPutResponseHopLimit": 1,
                "HttpEndpoint": "disabled"
            }
        }
    ]
}
```

Example 2: To describe a specified number of launch configurations

To return a specific number of launch configurations, use the `--max-items` option.

```
aws autoscaling describe-launch-configurations \
    --max-items 1
```

If the output includes a `NextToken` field, there are more launch configurations. To get the additional launch configurations, use the value of this field with the `--starting-token` option in a subsequent call as follows.

```
aws autoscaling describe-launch-configurations \
    --starting-token Z3M3LMPEXAMPLE
```

- For API details, see DescribeLaunchConfigurations in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the names of your launch configurations.

```
Get-ASLaunchConfiguration | format-table -property LaunchConfigurationName
```

Output:

```
LaunchConfigurationName
```

```
-----------------------
my-lc-1
my-lc-2
my-lc-3
my-lc-4
my-lc-5
```

Example 2: This example describes the specified launch configuration.

```
Get-ASLaunchConfiguration -LaunchConfigurationName my-lc-1
```

Output:

```
AssociatePublicIpAddress     : True
BlockDeviceMappings          : {/dev/xvda}
ClassicLinkVPCId             :
ClassicLinkVPCSecurityGroups : {}
CreatedTime                  : 12/12/2014 3:22:08 PM
EbsOptimized                 : False
IamInstanceProfile           :
ImageId                      : ami-043a5034
InstanceMonitoring           : Amazon.AutoScaling.Model.InstanceMonitoring
InstanceType                 : t2.micro
KernelId                     :
KeyName                      :
LaunchConfigurationARN       : arn:aws:autoscaling:us-
west-2:123456789012:launchConfiguration:7e5f31e4-693b-4604-9322-
                               e6f68d7fafad:launchConfigurationName/my-lc-1
LaunchConfigurationName      : my-lc-1
PlacementTenancy             :
RamdiskId                    :
SecurityGroups               : {sg-67ef0308}
SpotPrice                    :
UserData                     :
```

Example 3: This example describes the specified two launch configurations.

```
Get-ASLaunchConfiguration -LaunchConfigurationName @("my-lc-1", "my-lc-2")
```

Example 4: This example describes all your launch configurations.

```
Get-ASLaunchConfiguration
```

Example 5: This example describes all your launch configurations, in batches of 10.

```
$nextToken = $null
do {
  Get-ASLaunchConfiguration -NextToken $nextToken -MaxRecord 10
  $nextToken = $AWSHistory.LastServiceResponse.NextToken
} while ($nextToken -ne $null)
```

- For API details, see DescribeLaunchConfigurations in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeLifecycleHookTypes` with an AWS SDK or CLI

The following code examples show how to use `DescribeLifecycleHookTypes`.

CLI

AWS CLI

To describe the available lifecycle hook types

This example describes the available lifecycle hook types.

```
aws autoscaling describe-lifecycle-hook-types
```

Output:

```
{
    "LifecycleHookTypes": [
        "autoscaling:EC2_INSTANCE_LAUNCHING",
        "autoscaling:EC2_INSTANCE_TERMINATING"
    ]
}
```

- For API details, see DescribeLifecycleHookTypes in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the lifecycle hook types supported by Auto Scaling.

```
Get-ASLifecycleHookType
```

Output:

```
autoscaling:EC2_INSTANCE_LAUNCHING
auto-scaling:EC2_INSTANCE_TERMINATING
```

- For API details, see DescribeLifecycleHookTypes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeLifecycleHooks` with an AWS SDK or CLI

The following code examples show how to use `DescribeLifecycleHooks`.

CLI

AWS CLI

To describe your lifecycle hooks

This example describes the lifecycle hooks for the specified Auto Scaling group.

```
aws autoscaling describe-lifecycle-hooks \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "LifecycleHooks": [
        {
            "GlobalTimeout": 3000,
            "HeartbeatTimeout": 30,
            "AutoScalingGroupName": "my-asg",
            "LifecycleHookName": "my-launch-hook",
            "DefaultResult": "ABANDON",
            "LifecycleTransition": "autoscaling:EC2_INSTANCE_LAUNCHING"
        },
        {
            "GlobalTimeout": 6000,
            "HeartbeatTimeout": 60,
            "AutoScalingGroupName": "my-asg",
            "LifecycleHookName": "my-termination-hook",
            "DefaultResult": "CONTINUE",
            "LifecycleTransition": "autoscaling:EC2_INSTANCE_TERMINATING"
        }
    ]
}
```

- For API details, see DescribeLifecycleHooks in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the specified lifecycle hook.

```
Get-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName
 myLifecycleHook
```

Output:

```
AutoScalingGroupName  : my-asg
DefaultResult         : ABANDON
GlobalTimeout         : 172800
HeartbeatTimeout      : 3600
LifecycleHookName     : myLifecycleHook
LifecycleTransition   : auto-scaling:EC2_INSTANCE_LAUNCHING
NotificationMetadata  :
```

```
NotificationTargetARN : arn:aws:sns:us-west-2:123456789012:my-topic
RoleARN               : arn:aws:iam::123456789012:role/my-iam-role
```

Example 2: This example describes all lifecycle hooks for the specified Auto Scaling group.

```
Get-ASLifecycleHook -AutoScalingGroupName my-asg
```

Example 3: This example describes all lifecycle hooks for all your Auto Scaling groups.

```
Get-ASLifecycleHook
```

- For API details, see DescribeLifecycleHooks in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeLoadBalancers` with an AWS SDK or CLI

The following code examples show how to use `DescribeLoadBalancers`.

CLI

AWS CLI

To describe the Classic Load Balancers for an Auto Scaling group

This example describes the Classic Load Balancers for the specified Auto Scaling group.

```
aws autoscaling describe-load-balancers \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "LoadBalancers": [
        {
            "State": "Added",
```

```
            "LoadBalancerName": "my-load-balancer"
        }
    ]
}
```

- For API details, see DescribeLoadBalancers in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the load balancers for the specified Auto Scaling group.

```
Get-ASLoadBalancer -AutoScalingGroupName my-asg
```

Output:

```
LoadBalancerName    State
----------------    -----
my-lb               Added
```

- For API details, see DescribeLoadBalancers in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeMetricCollectionTypes` with an AWS SDK or CLI

The following code examples show how to use `DescribeMetricCollectionTypes`.

CLI

AWS CLI

To describe the available metric collection types

This example describes the available metric collection types.

```
aws autoscaling describe-metric-collection-types
```

Output:

```
{
    "Metrics": [
        {
            "Metric": "GroupMinSize"
        },
        {
            "Metric": "GroupMaxSize"
        },
        {
            "Metric": "GroupDesiredCapacity"
        },
        {
            "Metric": "GroupInServiceInstances"
        },
        {
            "Metric": "GroupInServiceCapacity"
        },
        {
            "Metric": "GroupPendingInstances"
        },
        {
            "Metric": "GroupPendingCapacity"
        },
        {
            "Metric": "GroupTerminatingInstances"
        },
        {
            "Metric": "GroupTerminatingCapacity"
        },
        {
            "Metric": "GroupStandbyInstances"
        },
        {
            "Metric": "GroupStandbyCapacity"
        },
        {
            "Metric": "GroupTotalInstances"
        },
        {
```

```
            "Metric": "GroupTotalCapacity"
        }
    ],
    "Granularities": [
        {
            "Granularity": "1Minute"
        }
    ]
}
```

For more information, see Auto Scaling group metrics in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeMetricCollectionTypes in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the metric collection types that are supported by Auto Scaling.

```
(Get-ASMetricCollectionType).Metrics
```

Output:

```
Metric
------
GroupMinSize
GroupMaxSize
GroupDesiredCapacity
GroupInServiceInstances
GroupPendingInstances
GroupTerminatingInstances
GroupStandbyInstances
GroupTotalInstances
```

Example 2: This example lists the corresponding granularities.

```
(Get-ASMetricCollectionType).Granularities
```

Output:

```
Granularity
-----------
1Minute
```

- For API details, see DescribeMetricCollectionTypes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeNotificationConfigurations` with an AWS SDK or CLI

The following code examples show how to use `DescribeNotificationConfigurations`.

CLI

AWS CLI

Example 1: To describe the notification configurations of a specified group

This example describes the notification configurations for the specified Auto Scaling group.

```
aws autoscaling describe-notification-configurations \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "NotificationConfigurations": [
        {
            "AutoScalingGroupName": "my-asg",
            "NotificationType": "autoscaling:TEST_NOTIFICATION",
            "TopicARN": "arn:aws:sns:us-west-2:123456789012:my-sns-topic-2"
        },
        {
            "AutoScalingGroupName": "my-asg",
            "NotificationType": "autoscaling:TEST_NOTIFICATION",
            "TopicARN": "arn:aws:sns:us-west-2:123456789012:my-sns-topic"
        }
    ]
```

```
}
```

For more information, see Getting Amazon SNS notifications when your Auto Scaling group scales in the *Amazon EC2 Auto Scaling User Guide*.

Example 1: To describe a specified number of notification configurations

To return a specific number of notification configurations, use the `max-items` parameter.

```
aws autoscaling describe-notification-configurations \
    --auto-scaling-group-name my-auto-scaling-group \
    --max-items 1
```

Output:

```
{
    "NotificationConfigurations": [
        {
            "AutoScalingGroupName": "my-asg",
            "NotificationType": "autoscaling:TEST_NOTIFICATION",
            "TopicARN": "arn:aws:sns:us-west-2:123456789012:my-sns-topic-2"
        },
        {
            "AutoScalingGroupName": "my-asg",
            "NotificationType": "autoscaling:TEST_NOTIFICATION",
            "TopicARN": "arn:aws:sns:us-west-2:123456789012:my-sns-topic"
        }
    ]
}
```

If the output includes a `NextToken` field, there are more notification configurations. To get the additional notification configurations, use the value of this field with the `starting-token` parameter in a subsequent call as follows.

```
aws autoscaling describe-notification-configurations \
    --auto-scaling-group-name my-asg \
    --starting-token Z3M3LMPEXAMPLE
```

For more information, see Getting Amazon SNS notifications when your Auto Scaling group scales in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeNotificationConfigurations in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the notification actions associated with the specified Auto Scaling group.

```
Get-ASNotificationConfiguration -AutoScalingGroupName my-asg | format-list
```

Output:

```
AutoScalingGroupName : my-asg
NotificationType     : auto-scaling:EC2_INSTANCE_LAUNCH
TopicARN             : arn:aws:sns:us-west-2:123456789012:my-topic

AutoScalingGroupName : my-asg
NotificationType     : auto-scaling:EC2_INSTANCE_TERMINATE
TopicARN             : arn:aws:sns:us-west-2:123456789012:my-topic
```

Example 2: This example describes the notification actions associated with all your Auto Scaling groups.

```
Get-ASNotificationConfiguration
```

- For API details, see DescribeNotificationConfigurations in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribePolicies` with an AWS SDK or CLI

The following code examples show how to use `DescribePolicies`.

CLI

AWS CLI

Example 1: To describe the scaling policies of a specified group

This example describes the scaling policies for the specified Auto Scaling group.

```
aws autoscaling describe-policies \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "ScalingPolicies": [
        {
            "AutoScalingGroupName": "my-asg",
            "PolicyName": "alb1000-target-tracking-scaling-policy",
            "PolicyARN": "arn:aws:autoscaling:us-
west-2:123456789012:scalingPolicy:3065d9c8-9969-4bec-
bb6a-3fbe5550fde6:autoScalingGroupName/my-asg:policyName/alb1000-target-tracking-
scaling-policy",
            "PolicyType": "TargetTrackingScaling",
            "StepAdjustments": [],
            "Alarms": [
                {
                    "AlarmName": "TargetTracking-my-asg-
AlarmHigh-924887a9-12d7-4e01-8686-6f844d13a196",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-
AlarmHigh-924887a9-12d7-4e01-8686-6f844d13a196"
                },
                {
                    "AlarmName": "TargetTracking-my-asg-AlarmLow-f96f899d-
b8e7-4d09-a010-c1aaa35da296",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-f96f899d-b8e7-4d09-a010-
c1aaa35da296"
                }
            ],
            "TargetTrackingConfiguration": {
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "ALBRequestCountPerTarget",
                    "ResourceLabel": "app/my-alb/778d41231b141a0f/targetgroup/my-
alb-target-group/943f017f100becff"
                },
                "TargetValue": 1000.0,
                "DisableScaleIn": false
            },
```

```
            "Enabled": true
        },
        {
            "AutoScalingGroupName": "my-asg",
            "PolicyName": "cpu40-target-tracking-scaling-policy",
            "PolicyARN": "arn:aws:autoscaling:us-
west-2:123456789012:scalingPolicy:5fd26f71-39d4-4690-82a9-
b8515c45cdde:autoScalingGroupName/my-asg:policyName/cpu40-target-tracking-
scaling-policy",
            "PolicyType": "TargetTrackingScaling",
            "StepAdjustments": [],
            "Alarms": [
                {
                    "AlarmName": "TargetTracking-my-asg-
AlarmHigh-139f9789-37b9-42ad-bea5-b5b147d7f473",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmHigh-139f9789-37b9-42ad-
bea5-b5b147d7f473"
                },
                {
                    "AlarmName": "TargetTracking-my-asg-AlarmLow-bd681c67-
fc18-4c56-8468-fb8e413009c9",
                    "AlarmARN": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:TargetTracking-my-asg-AlarmLow-bd681c67-fc18-4c56-8468-
fb8e413009c9"
                }
            ],
            "TargetTrackingConfiguration": {
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "ASGAverageCPUUtilization"
                },
                "TargetValue": 40.0,
                "DisableScaleIn": false
            },
            "Enabled": true
        }
    ]
}
```

For more information, see Dynamic scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To describe the scaling policies of a specified name

To return specific scaling policies, use the `--policy-names` option.

```
aws autoscaling describe-policies \
    --auto-scaling-group-name my-asg \
    --policy-names cpu40-target-tracking-scaling-policy
```

See example 1 for sample output.

For more information, see Dynamic scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To describe a number of scaling policies

To return a specific number of policies, use the `--max-items` option.

```
aws autoscaling describe-policies \
    --auto-scaling-group-name my-asg \
    --max-items 1
```

See example 1 for sample output.

If the output includes a `NextToken` field, use the value of this field with the `--starting-token` option in a subsequent call to get the additional policies.

```
aws autoscaling describe-policies --auto-scaling-group-name my-asg --starting-
token Z3M3LMPEXAMPLE
```

For more information, see Dynamic scaling in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribePolicies in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes all policies for the specified Auto Scaling group.

```
Get-ASPolicy -AutoScalingGroupName my-asg
```

Output:

```
AdjustmentType          : ChangeInCapacity
```

```
Alarms                  : {}
AutoScalingGroupName    : my-asg
Cooldown                : 0
EstimatedInstanceWarmup : 0
MetricAggregationType   :
MinAdjustmentMagnitude  : 0
MinAdjustmentStep       : 0
PolicyARN               : arn:aws:auto-scaling:us-
west-2:123456789012:scalingPolicy:aa3836ab-5462-42c7-adab-e1d769fc24ef
                          :autoScalingGroupName/my-asg:policyName/myScaleInPolicy
PolicyName              : myScaleInPolicy
PolicyType              : SimpleScaling
ScalingAdjustment       : -1
StepAdjustments         : {}
```

Example 2: This example describes the specified policies for the specified Auto Scaling group.

```
Get-ASPolicy -AutoScalingGroupName my-asg -PolicyName @("myScaleOutPolicy",
 "myScaleInPolicy")
```

Example 3: This example describes all policies for all your Auto Scaling groups.

```
Get-ASPolicy
```

- For API details, see DescribePolicies in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScalingActivities` with an AWS SDK or CLI

The following code examples show how to use `DescribeScalingActivities`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Amazon.AutoScaling.Model.Activity>>
 DescribeScalingActivitiesAsync(
        string groupName)
    {
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        return response.Activities;
    }
```

- For API details, see DescribeScalingActivities in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::DescribeScalingActivitiesRequest request;
        request.SetAutoScalingGroupName(groupName);

        Aws::Vector<Aws::AutoScaling::Model::Activity> allActivities;
        Aws::String nextToken; // Used for pagination;
        do {
            if (!nextToken.empty()) {
                request.SetNextToken(nextToken);
            }
            Aws::AutoScaling::Model::DescribeScalingActivitiesOutcome outcome =
                    autoScalingClient.DescribeScalingActivities(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::AutoScaling::Model::Activity> &activities
=
                        outcome.GetResult().GetActivities();
                allActivities.insert(allActivities.end(), activities.begin(),
activities.end());
                nextToken  = outcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error with AutoScaling::DescribeScalingActivities.
"
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
            }
        } while (!nextToken.empty());

        std::cout << "Found " << allActivities.size() << " activities."
                  << std::endl;
        std::cout << "Activities are ordered with the most recent first."
                  << std::endl;
        for (const Aws::AutoScaling::Model::Activity &activity: allActivities) {
            std::cout << activity.GetDescription() << std::endl;
            std::cout << activity.GetDetails() << std::endl;
        }
```

- For API details, see DescribeScalingActivities in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To describe scaling activities for the specified group

This example describes the scaling activities for the specified Auto Scaling group.

```
aws autoscaling describe-scaling-activities \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "Activities": [
        {
            "ActivityId": "f9f2d65b-f1f2-43e7-b46d-d86756459699",
            "Description": "Launching a new EC2 instance: i-0d4442563032606 0f",
            "AutoScalingGroupName": "my-asg",
            "Cause": "At 2020-10-30T19:35:51Z a user request update of
 AutoScalingGroup constraints to min: 0, max: 16, desired: 16 changing the
 desired capacity from 0 to 16.  At 2020-10-30T19:36:07Z an instance was started
 in response to a difference between desired and actual capacity, increasing the
 capacity from 0 to 16.",
            "StartTime": "2020-10-30T19:36:09.766Z",
            "EndTime": "2020-10-30T19:36:41Z",
            "StatusCode": "Successful",
```

```
            "Progress": 100,
            "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":
\"us-west-2b\"}"
        }
    ]
}
```

For more information, see Verify a scaling activity for an Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To describe the scaling activities for a deleted group

To describe scaling activities after the Auto Scaling group has been deleted, add the `--include-deleted-groups` option.

```
aws autoscaling describe-scaling-activities \
    --auto-scaling-group-name my-asg \
    --include-deleted-groups
```

Output:

```
{
    "Activities": [
        {
            "ActivityId": "e1f5de0e-f93e-1417-34ac-092a76fba220",
            "Description": "Launching a new EC2 instance.  Status Reason: Your
 Spot request price of 0.001 is lower than the minimum required Spot request
 fulfillment price of 0.0031. Launching EC2 instance failed.",
            "AutoScalingGroupName": "my-asg",
            "Cause": "At 2021-01-13T20:47:24Z a user request update of
 AutoScalingGroup constraints to min: 1, max: 5, desired: 3 changing the desired
 capacity from 0 to 3.  At 2021-01-13T20:47:27Z an instance was started in
 response to a difference between desired and actual capacity, increasing the
 capacity from 0 to 3.",
            "StartTime": "2021-01-13T20:47:30.094Z",
            "EndTime": "2021-01-13T20:47:30Z",
            "StatusCode": "Failed",
            "StatusMessage": "Your Spot request price of 0.001 is lower than
 the minimum required Spot request fulfillment price of 0.0031. Launching EC2
 instance failed.",
            "Progress": 100,
            "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":
\"us-west-2b\"}",
```

```
            "AutoScalingGroupState": "Deleted",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:283179a2-
f3ce-423d-93f6-66bb518232f7:autoScalingGroupName/my-asg"
        }
    ]
}
```

For more information, see Troubleshoot Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To describe a specified number of scaling activities

To return a specific number of activities, use the `--max-items` option.

```
aws autoscaling describe-scaling-activities \
    --max-items 1
```

Output:

```
{
    "Activities": [
        {
            "ActivityId": "f9f2d65b-f1f2-43e7-b46d-d86756459699",
            "Description": "Launching a new EC2 instance: i-0d44425630326060f",
            "AutoScalingGroupName": "my-asg",
            "Cause": "At 2020-10-30T19:35:51Z a user request update of
 AutoScalingGroup constraints to min: 0, max: 16, desired: 16 changing the
 desired capacity from 0 to 16.  At 2020-10-30T19:36:07Z an instance was started
 in response to a difference between desired and actual capacity, increasing the
 capacity from 0 to 16.",
            "StartTime": "2020-10-30T19:36:09.766Z",
            "EndTime": "2020-10-30T19:36:41Z",
            "StatusCode": "Successful",
            "Progress": 100,
            "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":
\"us-west-2b\"}"
        }
    ]
}
```

If the output includes a `NextToken` field, there are more activities. To get the additional activities, use the value of this field with the `--starting-token` option in a subsequent call as follows.

```
aws autoscaling describe-scaling-activities \
    --starting-token Z3M3LMPEXAMPLE
```

For more information, see Verify a scaling activity for an Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeScalingActivities in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void describeScalingActivities(AutoScalingClient
autoScalingClient, String groupName) {
       try {
           DescribeScalingActivitiesRequest scalingActivitiesRequest =
DescribeScalingActivitiesRequest.builder()
                   .autoScalingGroupName(groupName)
                   .maxRecords(10)
                   .build();

           DescribeScalingActivitiesResponse response = autoScalingClient
                   .describeScalingActivities(scalingActivitiesRequest);
           List<Activity> activities = response.activities();
           for (Activity activity : activities) {
               System.out.println("The activity Id is " +
activity.activityId());
               System.out.println("The activity details are " +
activity.details());
           }
```

```
        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see DescribeScalingActivities in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun describeAutoScalingGroups(groupName: String) {
    val groupsReques =
        DescribeAutoScalingGroupsRequest {
            autoScalingGroupNames = listOf(groupName)
            maxRecords = 10
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        val response = autoScalingClient.describeAutoScalingGroups(groupsReques)
        response.autoScalingGroups?.forEach { group ->
            println("The service to use for the health checks:
 ${group.healthCheckType}")
        }
    }
}
```

- For API details, see DescribeScalingActivities in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function describeScalingActivities($autoScalingGroupName)
    {
        return $this->autoScalingClient->describeScalingActivities([
            'AutoScalingGroupName' => $autoScalingGroupName,
        ]);
    }
```

- For API details, see DescribeScalingActivities in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the scaling activities for the last six weeks for the specified Auto Scaling group.

```
Get-ASScalingActivity -AutoScalingGroupName my-asg
```

Output:

```
ActivityId           : 063308ae-aa22-4a9b-94f4-9fae4EXAMPLE
AutoScalingGroupName : my-asg
Cause                : At 2015-11-22T15:45:16Z a user request explicitly set
 group desired capacity changing the desired
                       capacity from 1 to 2.  At 2015-11-22T15:45:34Z an instance
 was started in response to a difference
                       between desired and actual capacity, increasing the
 capacity from 1 to 2.
Description          : Launching a new EC2 instance: i-26e715fc
```

```
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              : 11/22/2015 7:46:09 AM
Progress             : 100
StartTime            : 11/22/2015 7:45:35 AM
StatusCode           : Successful
StatusMessage        :

ActivityId           : ce719997-086d-4c73-a2f1-ab703EXAMPLE
AutoScalingGroupName : my-asg
Cause                : At 2015-11-20T22:57:53Z a user request created an
 AutoScalingGroup changing the desired capacity
                        from 0 to 1.  At 2015-11-20T22:57:58Z an instance was
 started in response to a difference betwe
                       en desired and actual capacity, increasing the capacity
 from 0 to 1.
Description          : Launching a new EC2 instance: i-93633f9b
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              : 11/20/2015 2:58:32 PM
Progress             : 100
StartTime            : 11/20/2015 2:57:59 PM
StatusCode           : Successful
StatusMessage        :
```

Example 2: This example describes the specified scaling activity.

```
Get-ASScalingActivity -ActivityId "063308ae-aa22-4a9b-94f4-9fae4EXAMPLE"
```

Example 3: This example describes the scaling activities for the last six weeks for all your Auto Scaling groups.

```
Get-ASScalingActivity
```

- For API details, see DescribeScalingActivities in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def describe_scaling_activities(self, group_name: str) -> List[Dict[str,
 Any]]:
        """
        Gets information about scaling activities for the group. Scaling
 activities
        are things like instances stopping or starting in response to user
 requests
        or capacity changes.

        :param group_name: The name of the group to look up.
        :return: A list of dictionaries representing the scaling activities for
 the
                 group, ordered with the most recent activity first.
        :raises ClientError: If there is an error describing the scaling
 activities.
        """
        try:
            paginator = self.autoscaling_client.get_paginator(
                "describe_scaling_activities"
            )
            response_iterator =
 paginator.paginate(AutoScalingGroupName=group_name)
```

```
            activities = []
            for response in response_iterator:
                activities.extend(response.get("Activities", []))

            logger.info(
                f"Successfully described scaling activities for group
'{group_name}'."
            )

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(
                f"Couldn't describe scaling activities for group '{group_name}'.
Error code: {error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "ResourceContentionFault":
                logger.error(
                    f"There is a conflict with another operation that is
modifying the Auto Scaling group '{group_name}'. "
                    "Please try again later."
                )
            raise
        else:
            return activities
```

- For API details, see DescribeScalingActivities in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    pub async fn describe_scenario(&self) -> AutoScalingScenarioDescription {
        let group = self
```

```
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await
            .map(|s| {
                s.auto_scaling_groups()
                    .iter()
                    .map(|s| {
                        format!(
                            "{}: {}",
                            s.auto_scaling_group_name().unwrap_or("Unknown"),
                            s.status().unwrap_or("Unknown")
                        )
                    })
                    .collect::<Vec<String>>()
            })
            .map_err(|e| {
                ScenarioError::new("Failed to describe auto scaling groups for
scenario", &e)
            });

        let instances = self
            .list_instances()
            .await
            .map_err(|e| anyhow!("There was an error listing instances: {e}",));

        // 10. DescribeScalingActivities: list the scaling activities that have
occurred for the group so far.
        //   Bonus: use CloudWatch API to get and show some metrics collected for
the group.
        //   CW.ListMetrics with Namespace='AWS/AutoScaling' and
Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
        //   CW.GetMetricStatistics with Statistics='Sum'. Start and End times
must be in UTC!
        let activities = self
            .autoscaling
            .describe_scaling_activities()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .into_paginator()
            .items()
            .send()
            .collect::<Result<Vec<_>, _>>()
            .await
```

```
            .map_err(|e| {
                anyhow!(
                    "There was an error retrieving scaling activities: {}",
                    DisplayErrorContext(&e)
                )
            });

        AutoScalingScenarioDescription {
            group,
            instances,
            activities,
        }
    }
```

- For API details, see DescribeScalingActivities in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScalingProcessTypes` with an AWS SDK or CLI

The following code examples show how to use `DescribeScalingProcessTypes`.

CLI

AWS CLI

To describe the available process types

This example describes the available process types.

```
aws autoscaling describe-scaling-process-types
```

Output:

```
{
    "Processes": [
        {
            "ProcessName": "AZRebalance"
```

```
        },
        {
            "ProcessName": "AddToLoadBalancer"
        },
        {
            "ProcessName": "AlarmNotification"
        },
        {
            "ProcessName": "HealthCheck"
        },
        {
            "ProcessName": "InstanceRefresh"
        },
        {
            "ProcessName": "Launch"
        },
        {
            "ProcessName": "ReplaceUnhealthy"
        },
        {
            "ProcessName": "ScheduledActions"
        },
        {
            "ProcessName": "Terminate"
        }
    ]
}
```

For more information, see Suspending and resuming scaling processes in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeScalingProcessTypes in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the process types that are supported by Auto Scaling.

```
Get-ASScalingProcessType
```

Output:

```
ProcessName
-----------
AZRebalance
AddToLoadBalancer
AlarmNotification
HealthCheck
Launch
ReplaceUnhealthy
ScheduledActions
Terminate
```

- For API details, see DescribeScalingProcessTypes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeScheduledActions` with an AWS SDK or CLI

The following code examples show how to use `DescribeScheduledActions`.

CLI

AWS CLI

Example 1: To describe all scheduled actions

This example describes all your scheduled actions.

```
aws autoscaling describe-scheduled-actions
```

Output:

```
{
    "ScheduledUpdateGroupActions": [
        {
            "AutoScalingGroupName": "my-asg",
            "ScheduledActionName": "my-recurring-action",
            "Recurrence": "30 0 1 1,6,12 *",
```

```
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-
action",
            "StartTime": "2023-12-01T04:00:00Z",
            "Time": "2023-12-01T04:00:00Z",
            "MinSize": 1,
            "MaxSize": 6,
            "DesiredCapacity": 4,
            "TimeZone": "America/New_York"
        }
    ]
}
```

For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To describe scheduled actions for the specified group

To describe the scheduled actions for a specific Auto Scaling group, use the `--auto-scaling-group-name` option.

```
aws autoscaling describe-scheduled-actions \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "ScheduledUpdateGroupActions": [
        {
            "AutoScalingGroupName": "my-asg",
            "ScheduledActionName": "my-recurring-action",
            "Recurrence": "30 0 1 1,6,12 *",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-
action",
            "StartTime": "2023-12-01T04:00:00Z",
            "Time": "2023-12-01T04:00:00Z",
            "MinSize": 1,
            "MaxSize": 6,
            "DesiredCapacity": 4,
            "TimeZone": "America/New_York"
```

```
        }
    ]
}
```

For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To describe the specified scheduled action

To describe a specific scheduled action, use the `--scheduled-action-names` option.

```
aws autoscaling describe-scheduled-actions \
    --scheduled-action-names my-recurring-action
```

Output:

```
{
    "ScheduledUpdateGroupActions": [
        {
            "AutoScalingGroupName": "my-asg",
            "ScheduledActionName": "my-recurring-action",
            "Recurrence": "30 0 1 1,6,12 *",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-
action",
            "StartTime": "2023-12-01T04:00:00Z",
            "Time": "2023-12-01T04:00:00Z",
            "MinSize": 1,
            "MaxSize": 6,
            "DesiredCapacity": 4,
            "TimeZone": "America/New_York"
        }
    ]
}
```

For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 4: To describe scheduled actions with a specified start time

To describe the scheduled actions that start at a specific time, use the `--start-time` option.

```
aws autoscaling describe-scheduled-actions \
    --start-time "2023-12-01T04:00:00Z"
```

Output:

```
{
    "ScheduledUpdateGroupActions": [
        {
            "AutoScalingGroupName": "my-asg",
            "ScheduledActionName": "my-recurring-action",
            "Recurrence": "30 0 1 1,6,12 *",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-
action",
            "StartTime": "2023-12-01T04:00:00Z",
            "Time": "2023-12-01T04:00:00Z",
            "MinSize": 1,
            "MaxSize": 6,
            "DesiredCapacity": 4,
            "TimeZone": "America/New_York"
        }
    ]
}
```

For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 5: To describe scheduled actions that end at a specified time

To describe the scheduled actions that end at a specific time, use the `--end-time` option.

```
aws autoscaling describe-scheduled-actions \
    --end-time "2023-12-01T04:00:00Z"
```

Output:

```
{
    "ScheduledUpdateGroupActions": [
        {
            "AutoScalingGroupName": "my-asg",
            "ScheduledActionName": "my-recurring-action",
```

```
            "Recurrence": "30 0 1 1,6,12 *",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-
action",
            "StartTime": "2023-12-01T04:00:00Z",
            "Time": "2023-12-01T04:00:00Z",
            "MinSize": 1,
            "MaxSize": 6,
            "DesiredCapacity": 4,
            "TimeZone": "America/New_York"
        }
    ]
}
```

For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 6: To describe a specified number of scheduled actions

To return a specific number of scheduled actions, use the `--max-items` option.

```
aws autoscaling describe-scheduled-actions \
    --auto-scaling-group-name my-asg \
    --max-items 1
```

Output:

```
{
    "ScheduledUpdateGroupActions": [
        {
            "AutoScalingGroupName": "my-asg",
            "ScheduledActionName": "my-recurring-action",
            "Recurrence": "30 0 1 1,6,12 *",
            "ScheduledActionARN": "arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8e86b655-b2e6-4410-8f29-
b4f094d6871c:autoScalingGroupName/my-asg:scheduledActionName/my-recurring-
action",
            "StartTime": "2023-12-01T04:00:00Z",
            "Time": "2023-12-01T04:00:00Z",
            "MinSize": 1,
            "MaxSize": 6,
            "DesiredCapacity": 4,
            "TimeZone": "America/New_York"
```

```
        }
    ]
}
```

If the output includes a `NextToken` field, there are more scheduled actions. To get the additional scheduled actions, use the value of this field with the `--starting-token` option in a subsequent call as follows.

```
aws autoscaling describe-scheduled-actions \
    --auto-scaling-group-name my-asg \
    --starting-token Z3M3LMPEXAMPLE
```

For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeScheduledActions in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the scheduled scaling actions for the specified Auto Scaling group.

```
Get-ASScheduledAction -AutoScalingGroupName my-asg
```

Output:

```
AutoScalingGroupName : my-asg
DesiredCapacity      : 10
EndTime              :
MaxSize              :
MinSize              :
Recurrence           :
ScheduledActionARN   : arn:aws:autoscaling:us-
west-2:123456789012:scheduledUpdateGroupAction:8a4c5f24-6ec6-4306-a2dd-f7
                       2c3af3a4d6:autoScalingGroupName/my-
asg:scheduledActionName/myScheduledAction
ScheduledActionName  : myScheduledAction
StartTime            : 11/30/2015 8:00:00 AM
Time                 : 11/30/2015 8:00:00 AM
```

Example 2: This example describes the specified scheduled scaling actions.

```
Get-ASScheduledAction -ScheduledActionName @("myScheduledScaleOut",
 "myScheduledScaleIn")
```

Example 3: This example describes the scheduled scaling actions that start by the specified time.

```
Get-ASScheduledAction -StartTime "2015-12-01T08:00:00Z"
```

Example 4: This example describes the scheduled scaling actions that end by the specified time.

```
Get-ASScheduledAction -EndTime "2015-12-30T08:00:00Z"
```

Example 5: This example describes the scheduled scaling actions for all your Auto Scaling groups.

```
Get-ASScheduledAction
```

- For API details, see DescribeScheduledActions in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeTags` with an AWS SDK or CLI

The following code examples show how to use `DescribeTags`.

CLI

AWS CLI

To describe all tags

This example describes all your tags.

```
aws autoscaling describe-tags
```

Output:

```
{
    "Tags": [
        {
            "ResourceType": "auto-scaling-group",
            "ResourceId": "my-asg",
            "PropagateAtLaunch": true,
            "Value": "Research",
            "Key": "Dept"
        },
        {
            "ResourceType": "auto-scaling-group",
            "ResourceId": "my-asg",
            "PropagateAtLaunch": true,
            "Value": "WebServer",
            "Key": "Role"
        }
    ]
}
```

For more information, see Tagging Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To describe tags for a specified group

To describe tags for a specific Auto Scaling group, use the `--filters` option.

```
aws autoscaling describe-tags --filters Name=auto-scaling-group,Values=my-asg
```

For more information, see Tagging Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To describe the specified number of tags

To return a specific number of tags, use the `--max-items` option.

```
aws autoscaling describe-tags \
```

```
    --max-items 1
```

If the output includes a `NextToken` field, there are more tags. To get the additional tags, use the value of this field with the `--starting-token` option in a subsequent call as follows.

```
aws autoscaling describe-tags \
    --filters Name=auto-scaling-group,Values=my-asg \
    --starting-token Z3M3LMPEXAMPLE
```

For more information, see Tagging Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeTags in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example describes the tags with a key value of either 'myTag' or 'myTag2'. The possible values for the filter name are 'auto-scaling-group', 'key', 'value', and 'propagate-at-launch'. The syntax used by this example requires PowerShell version 3 or later.

```
Get-ASTag -Filter @( @{ Name="key"; Values=@("myTag", "myTag2") } )
```

Output:

```
Key               : myTag2
PropagateAtLaunch : True
ResourceId        : my-asg
ResourceType      : auto-scaling-group
Value             : myTagValue2

Key               : myTag
PropagateAtLaunch : True
ResourceId        : my-asg
ResourceType      : auto-scaling-group
Value             : myTagValue
```

Example 2: With PowerShell version 2, you must use New-Object to create the filter for the Filter parameter.

```
$keys = New-Object string[] 2
$keys[0] = "myTag"
$keys[1] = "myTag2"
$filter = New-Object Amazon.AutoScaling.Model.Filter
$filter.Name = "key"
$filter.Values = $keys
Get-ASTag -Filter @( $filter )
```

Example 3: This example describes all tags for all your Auto Scaling groups.

```
Get-ASTag
```

- For API details, see DescribeTags in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeTerminationPolicyTypes` with an AWS SDK or CLI

The following code examples show how to use `DescribeTerminationPolicyTypes`.

CLI

AWS CLI

To describe available termination policy types

This example describes the available termination policy types.

```
aws autoscaling describe-termination-policy-types
```

Output:

```
{
    "TerminationPolicyTypes": [
        "AllocationStrategy",
        "ClosestToNextInstanceHour",
```

```
        "Default",
        "NewestInstance",
        "OldestInstance",
        "OldestLaunchConfiguration",
        "OldestLaunchTemplate"
    ]
}
```

For more information, see Controlling which Auto Scaling instances terminate during scale in in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DescribeTerminationPolicyTypes in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the termination policies that are supported by Auto Scaling.

```
Get-ASTerminationPolicyType
```

Output:

```
ClosestToNextInstanceHour
Default
NewestInstance
OldestInstance
OldestLaunchConfiguration
```

- For API details, see DescribeTerminationPolicyTypes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DetachInstances` with an AWS SDK or CLI

The following code examples show how to use `DetachInstances`.

CLI

AWS CLI

To detach an instance from an Auto Scaling group

This example detaches the specified instance from the specified Auto Scaling group.

```
aws autoscaling detach-instances \
    --instance-ids i-030017cfa84b20135 \
    --auto-scaling-group-name my-asg \
    --should-decrement-desired-capacity
```

Output:

```
{
    "Activities": [
        {
            "ActivityId": "5091cb52-547a-47ce-a236-c9ccbc2cb2c9",
            "AutoScalingGroupName": "my-asg",
            "Description": "Detaching EC2 instance: i-030017cfa84b20135",
            "Cause": "At 2020-10-31T17:35:04Z instance i-030017cfa84b20135 was
 detached in response to a user request, shrinking the capacity from 2 to 1.",
            "StartTime": "2020-04-12T15:02:16.179Z",
            "StatusCode": "InProgress",
            "Progress": 50,
            "Details": "{\"Subnet ID\":\"subnet-6194ea3b\",\"Availability Zone\":
\"us-west-2c\"}"
        }
    ]
}
```

- For API details, see DetachInstances in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example detaches the specified instance from the specified Auto Scaling group and decreases the desired capacity so that Auto Scaling does not launch a replacement instance.

```
Dismount-ASInstance -InstanceId i-93633f9b -AutoScalingGroupName my-asg -
ShouldDecrementDesiredCapacity $true
```

Output:

```
ActivityId           : 06733445-ce94-4039-be1b-b9f1866e276e
AutoScalingGroupName : my-asg
Cause                : At 2015-11-20T22:34:59Z instance i-93633f9b was detached
 in response to a user request, shrinking
                       the capacity from 2 to 1.
Description          : Detaching EC2 instance: i-93633f9b
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              :
Progress             : 50
StartTime            : 11/20/2015 2:34:59 PM
StatusCode           : InProgress
StatusMessage        :
```

Example 2: This example detaches the specified instance from the specified Auto Scaling group without decreasing the desired capacity. Auto Scaling launches a replacement instance.

```
Dismount-ASInstance -InstanceId i-7bf746a2 -AutoScalingGroupName my-asg -
ShouldDecrementDesiredCapacity $false
```

Output:

```
ActivityId           : f43a3cd4-d38c-4af7-9fe0-d76ec2307b6d
AutoScalingGroupName : my-asg
Cause                : At 2015-11-20T22:34:59Z instance i-7bf746a2 was detached
 in response to a user request.
Description          : Detaching EC2 instance: i-7bf746a2
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              :
Progress             : 50
StartTime            : 11/20/2015 2:34:59 PM
StatusCode           : InProgress
StatusMessage        :
```

- For API details, see DetachInstances in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DetachLoadBalancers` with an AWS SDK or CLI

The following code examples show how to use `DetachLoadBalancers`.

CLI

AWS CLI

To detach a Classic Load Balancer from an Auto Scaling group

This example detaches the specified Classic Load Balancer from the specified Auto Scaling group.

```
aws autoscaling detach-load-balancers \
    --load-balancer-names my-load-balancer \
    --auto-scaling-group-name my-asg
```

This command produces no output.

For more information, see Attaching a load balancer to your Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DetachLoadBalancers in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example detaches the specified load balancer from the specified Auto Scaling group.

```
Dismount-ASLoadBalancer -LoadBalancerName my-lb -AutoScalingGroupName my-asg
```

- For API details, see DetachLoadBalancers in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DisableMetricsCollection` with an AWS SDK or CLI

The following code examples show how to use `DisableMetricsCollection`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DisableMetricsCollection in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::DisableMetricsCollectionRequest request;
        request.SetAutoScalingGroupName(groupName);

        Aws::AutoScaling::Model::DisableMetricsCollectionOutcome outcome =
                autoScalingClient.DisableMetricsCollection(request);

        if (outcome.IsSuccess()) {
            std::cout << "Metrics collection has been disabled." << std::endl;
        }
        else {
            std::cerr << "Error with AutoScaling::DisableMetricsCollection. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

        }
```

- For API details, see DisableMetricsCollection in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To disable metrics collection for an Auto Scaling group

This example disables collection of the `GroupDesiredCapacity` metric for the specified Auto Scaling group.

```
aws autoscaling disable-metrics-collection \
    --auto-scaling-group-name my-asg \
    --metrics GroupDesiredCapacity
```

This command produces no output.

For more information, see Monitoring CloudWatch metrics for your Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see DisableMetricsCollection in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void disableMetricsCollection(AutoScalingClient
autoScalingClient, String groupName) {
       try {
           DisableMetricsCollectionRequest disableMetricsCollectionRequest =
DisableMetricsCollectionRequest.builder()
                   .autoScalingGroupName(groupName)
                   .metrics("GroupMaxSize")
                   .build();


autoScalingClient.disableMetricsCollection(disableMetricsCollectionRequest);
```

```
            System.out.println("The disable metrics collection operation was
 successful");

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see DisableMetricsCollection in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun disableMetricsCollection(groupName: String) {
    val disableMetricsCollectionRequest =
        DisableMetricsCollectionRequest {
            autoScalingGroupName = groupName
            metrics = listOf("GroupMaxSize")
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->

 autoScalingClient.disableMetricsCollection(disableMetricsCollectionRequest)
        println("The disable metrics collection operation was successful")
    }
}
```

- For API details, see DisableMetricsCollection in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function disableMetricsCollection($autoScalingGroupName)
    {
        return $this->autoScalingClient->disableMetricsCollection([
            'AutoScalingGroupName' => $autoScalingGroupName,
        ]);
    }
```

- For API details, see DisableMetricsCollection in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example disables monitoring of the specified metrics for the specified Auto Scaling group.

```
Disable-ASMetricsCollection -AutoScalingGroupName my-asg -Metric
 @("GroupMinSize", "GroupMaxSize")
```

Example 2: This example disables monitoring of all metrics for the specified Auto Scaling group.

```
Disable-ASMetricsCollection -AutoScalingGroupName my-asg
```

- For API details, see DisableMetricsCollection in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def disable_metrics(self, group_name: str) -> Dict[str, Any]:
        """
        Stops CloudWatch metric collection for the Auto Scaling group.

        :param group_name: The name of the group.
        :return: A dictionary with the response from disabling the metrics
 collection.
        :raises ClientError: If there is an error disabling metrics collection.
        """
        try:
            response = self.autoscaling_client.disable_metrics_collection(
                AutoScalingGroupName=group_name
            )
            logger.info(
                f"Successfully disabled metrics collection for group
 '{group_name}'."
            )
            return response
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(
```

```
                f"Couldn't disable metrics for group '{group_name}'. Error code:
{error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "ResourceContentionFault":
                logger.error(
                    f"There is a conflict with another operation that is
modifying the Auto Scaling group '{group_name}'. "
                    "Please try again later."
                )
            raise
```

- For API details, see DisableMetricsCollection in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        // If this fails it's fine, just means there are extra cloudwatch metrics
events for the scale-down.
        let _ = self
            .autoscaling
            .disable_metrics_collection()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .send()
            .await;
```

- For API details, see DisableMetricsCollection in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `EnableMetricsCollection` with an AWS SDK or CLI

The following code examples show how to use `EnableMetricsCollection`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Enable the collection of metric data for an Auto Scaling group.
/// </summary>
/// <param name="groupName">The name of the Auto Scaling group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> EnableMetricsCollectionAsync(string groupName)
{
    var listMetrics = new List<string>
        {
            "GroupMaxSize",
        };

    var collectionRequest = new EnableMetricsCollectionRequest
    {
        AutoScalingGroupName = groupName,
        Metrics = listMetrics,
        Granularity = "1Minute",
    };
```

```
        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see EnableMetricsCollection in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::EnableMetricsCollectionRequest request;
        request.SetAutoScalingGroupName(groupName);

        request.AddMetrics("GroupMinSize");
        request.AddMetrics("GroupMaxSize");
        request.AddMetrics("GroupDesiredCapacity");
        request.AddMetrics("GroupInServiceInstances");
        request.AddMetrics("GroupTotalInstances");
        request.SetGranularity("1Minute");

        Aws::AutoScaling::Model::EnableMetricsCollectionOutcome outcome =
                autoScalingClient.EnableMetricsCollection(request);
        if (outcome.IsSuccess()) {
            std::cout << "Auto Scaling metrics have been enabled."
                      << std::endl;
        }
```

```
        else {
            std::cerr << "Error with AutoScaling::EnableMetricsCollection. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

        }
```

- For API details, see EnableMetricsCollection in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To enable metrics collection for an Auto Scaling group

This example enables data collection for the specified Auto Scaling group.

```
aws autoscaling enable-metrics-collection \
    --auto-scaling-group-name my-asg \
    --granularity "1Minute"
```

This command produces no output.

For more information, see Monitoring CloudWatch metrics for your Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To collect data for the specified metric for an Auto Scaling group

To collect data for a specific metric, use the `--metrics` option.

```
aws autoscaling enable-metrics-collection \
    --auto-scaling-group-name my-asg \
    --metrics GroupDesiredCapacity --granularity "1Minute"
```

This command produces no output.

For more information, see Monitoring CloudWatch metrics for your Auto Scaling groups and instances in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see EnableMetricsCollection in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void enableMetricsCollection(AutoScalingClient
autoScalingClient, String groupName) {
       try {
           EnableMetricsCollectionRequest collectionRequest =
EnableMetricsCollectionRequest.builder()
                   .autoScalingGroupName(groupName)
                   .metrics("GroupMaxSize")
                   .granularity("1Minute")
                   .build();

           autoScalingClient.enableMetricsCollection(collectionRequest);
           System.out.println("The enable metrics collection operation was
successful");

       } catch (AutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- For API details, see EnableMetricsCollection in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun enableMetricsCollection(groupName: String?) {
    val collectionRequest =
        EnableMetricsCollectionRequest {
            autoScalingGroupName = groupName
            metrics = listOf("GroupMaxSize")
            granularity = "1Minute"
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.enableMetricsCollection(collectionRequest)
        println("The enable metrics collection operation was successful")
    }
}
```

- For API details, see EnableMetricsCollection in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function enableMetricsCollection($autoScalingGroupName, $granularity)
    {
        return $this->autoScalingClient->enableMetricsCollection([
```

```
            'AutoScalingGroupName' => $autoScalingGroupName,
            'Granularity' => $granularity,
        ]);
    }
```

- For API details, see EnableMetricsCollection in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example enables monitoring of the specified metrics for the specified Auto Scaling group.

```
Enable-ASMetricsCollection  -Metric @("GroupMinSize", "GroupMaxSize") -
AutoScalingGroupName my-asg -Granularity 1Minute
```

Example 2: This example enables monitoring of all metrics for the specified Auto Scaling group.

```
Enable-ASMetricsCollection -AutoScalingGroupName my-asg -Granularity 1Minute
```

- For API details, see EnableMetricsCollection in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
```

```
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def enable_metrics(self, group_name: str, metrics: List[str]) -> Dict[str,
Any]:
        """
        Enables CloudWatch metric collection for Amazon EC2 Auto Scaling
activities.

        :param group_name: The name of the group to enable.
        :param metrics: A list of metrics to collect.
        :return: A dictionary with the response from enabling the metrics
collection.
        :raises ClientError: If there is an error enabling metrics collection.
        """
        try:
            response = self.autoscaling_client.enable_metrics_collection(
                AutoScalingGroupName=group_name, Metrics=metrics,
Granularity="1Minute"
            )
            logger.info(
                f"Successfully enabled metrics for Auto Scaling group
'{group_name}'."
            )

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(
                f"Couldn't enable metrics on '{group_name}'. Error code:
{error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "ResourceContentionFault":
                logger.error(
                    f"There is a conflict with another operation that is
modifying the Auto Scaling group '{group_name}'. "
                    "Please try again later."
                )
            elif error_code == "InvalidParameterCombination":
                logger.error(
                    f"The combination of parameters provided for enabling metrics
on '{group_name}' is not valid. "
```

```
                    "Please check the parameters and try again."
                )
            raise
        else:
            return response
```

- For API details, see EnableMetricsCollection in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        let enable_metrics_collection = autoscaling
            .enable_metrics_collection()
            .auto_scaling_group_name(auto_scaling_group_name.as_str())
            .granularity("1Minute")
            .set_metrics(Some(vec![
                String::from("GroupMinSize"),
                String::from("GroupMaxSize"),
                String::from("GroupDesiredCapacity"),
                String::from("GroupInServiceInstances"),
                String::from("GroupTotalInstances"),
            ]))
            .send()
            .await;
```

- For API details, see EnableMetricsCollection in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `EnterStandby` with an AWS SDK or CLI

The following code examples show how to use `EnterStandby`.

CLI

AWS CLI

To move instances into standby mode

This example puts the specified instance into standby mode. This is useful for updating or troubleshooting an instance that is currently in service.

```
aws autoscaling enter-standby \
    --instance-ids i-061c63c5eb45f0416 \
    --auto-scaling-group-name my-asg \
    --should-decrement-desired-capacity
```

Output:

```
{
    "Activities": [
        {
            "ActivityId": "ffa056b4-6ed3-41ba-ae7c-249dfae6eba1",
            "AutoScalingGroupName": "my-asg",
            "Description": "Moving EC2 instance to Standby: i-061c63c5eb45f0416",
            "Cause": "At 2020-10-31T20:31:00Z instance i-061c63c5eb45f0416 was
 moved to standby in response to a user request, shrinking the capacity from 1 to
 0.",
            "StartTime": "2020-10-31T20:31:00.949Z",
            "StatusCode": "InProgress",
            "Progress": 50,
            "Details": "{\"Subnet ID\":\"subnet-6194ea3b\",\"Availability Zone\":
\"us-west-2c\"}"
        }
    ]
}
```

For more information, see Amazon EC2 Auto Scaling instance lifecycle in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see EnterStandby in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example puts the specified instance into standby mode and decreases the desired capacity so that Auto Scaling does not launch a replacement instance.

```
Enter-ASStandby -InstanceId i-93633f9b -AutoScalingGroupName my-asg -
ShouldDecrementDesiredCapacity $true
```

Output:

```
ActivityId           : e36a5a54-ced6-4df8-bd19-708e2a59a649
AutoScalingGroupName : my-asg
Cause                : At 2015-11-22T15:48:06Z instance i-95b8484f was moved to
 standby in response to a user request,
                       shrinking the capacity from 2 to 1.
Description          : Moving EC2 instance to Standby: i-95b8484f
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              :
Progress             : 50
StartTime            : 11/22/2015 7:48:06 AM
StatusCode           : InProgress
StatusMessage        :
```

Example 2: This example puts the specified instance into standby mode without decreasing the desired capacity. Auto Scaling launches a replacement instance.

```
Enter-ASStandby -InstanceId i-93633f9b -AutoScalingGroupName my-asg -
ShouldDecrementDesiredCapacity $false
```

Output:

```
ActivityId           : e36a5a54-ced6-4df8-bd19-708e2a59a649
AutoScalingGroupName : my-asg
Cause                : At 2015-11-22T15:48:06Z instance i-95b8484f was moved to
 standby in response to a user request.
Description          : Moving EC2 instance to Standby: i-95b8484f
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
```

```
EndTime              :
Progress             : 50
StartTime            : 11/22/2015 7:48:06 AM
StatusCode           : InProgress
StatusMessage        :
```

- For API details, see EnterStandby in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ExecutePolicy` with an AWS SDK or CLI

The following code examples show how to use `ExecutePolicy`.

CLI

AWS CLI

To execute a scaling policy

This example executes the scaling policy named `my-step-scale-out-policy` for the specified Auto Scaling group.

```
aws autoscaling execute-policy \
    --auto-scaling-group-name my-asg \
    --policy-name  my-step-scale-out-policy \
    --metric-value 95 \
    --breach-threshold 80
```

This command produces no output.

For more information, see Step and simple scaling policies in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see ExecutePolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example executes the specified policy for the specified Auto Scaling group.

```
Start-ASPolicy -AutoScalingGroupName my-asg -PolicyName "myScaleInPolicy"
```

Example 2: This example executes the specified policy for the specified Auto Scaling group, after waiting for the cooldown period to complete.

```
Start-ASPolicy -AutoScalingGroupName my-asg -PolicyName "myScaleInPolicy" -
HonorCooldown $true
```

- For API details, see ExecutePolicy in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ExitStandby` with an AWS SDK or CLI

The following code examples show how to use `ExitStandby`.

CLI

AWS CLI

To move instances out of standby mode

This example moves the specified instance out of standby mode.

```
aws autoscaling exit-standby \
    --instance-ids i-061c63c5eb45f0416 \
    --auto-scaling-group-name my-asg
```

Output:

```
{
    "Activities": [
```

```
        {
            "ActivityId": "142928e1-a2dc-453a-9b24-b85ad6735928",
            "AutoScalingGroupName": "my-asg",
            "Description": "Moving EC2 instance out of Standby:
 i-061c63c5eb45f0416",
            "Cause": "At 2020-10-31T20:32:50Z instance i-061c63c5eb45f0416 was
 moved out of standby in response to a user request, increasing the capacity from
 0 to 1.",
            "StartTime": "2020-10-31T20:32:50.222Z",
            "StatusCode": "PreInService",
            "Progress": 30,
            "Details": "{\"Subnet ID\":\"subnet-6194ea3b\",\"Availability Zone\":
\"us-west-2c\"}"
        }
    ]
}
```

For more information, see Temporarily removing instances from your Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see ExitStandby in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example moves the specified instance out of standby mode.

```
Exit-ASStandby -InstanceId i-93633f9b -AutoScalingGroupName my-asg
```

Output:

```
ActivityId           : 1833d3e8-e32f-454e-b731-0670ad4c6934
AutoScalingGroupName : my-asg
Cause                : At 2015-11-22T15:51:21Z instance i-95b8484f was moved out
 of standby in response to a user
                       request, increasing the capacity from 1 to 2.
Description          : Moving EC2 instance out of Standby: i-95b8484f
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              :
Progress             : 30
StartTime            : 11/22/2015 7:51:21 AM
```

```
StatusCode           : PreInService
StatusMessage        :
```

- For API details, see ExitStandby in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutLifecycleHook` with an AWS SDK or CLI

The following code examples show how to use `PutLifecycleHook`.

CLI

AWS CLI

Example 1: To create a lifecycle hook

This example creates a lifecycle hook that will invoke on any newly launched instances, with a timeout of 4800 seconds. This is useful for keeping the instances in a wait state until the user data scripts have finished, or for invoking an AWS Lambda function using EventBridge.

```
aws autoscaling put-lifecycle-hook \
    --auto-scaling-group-name my-asg \
    --lifecycle-hook-name my-launch-hook \
    --lifecycle-transition autoscaling:EC2_INSTANCE_LAUNCHING \
    --heartbeat-timeout 4800
```

This command produces no output. If a lifecycle hook with the same name already exists, it will be overwritten by the new lifecycle hook.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To send an Amazon SNS email message to notify you of instance state transitions

This example creates a lifecycle hook with the Amazon SNS topic and IAM role to use to receive notification at instance launch.

```
aws autoscaling put-lifecycle-hook \
```

```
    --auto-scaling-group-name my-asg \
    --lifecycle-hook-name my-launch-hook \
    --lifecycle-transition autoscaling:EC2_INSTANCE_LAUNCHING \
    --notification-target-arn arn:aws:sns:us-west-2:123456789012:my-sns-topic \
    --role-arn arn:aws:iam::123456789012:role/my-auto-scaling-role
```

This command produces no output.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To publish a message to an Amazon SQS queue

This example creates a lifecycle hook that publishes a message with metadata to the specified Amazon SQS queue.

```
aws autoscaling put-lifecycle-hook \
    --auto-scaling-group-name my-asg \
    --lifecycle-hook-name my-launch-hook \
    --lifecycle-transition autoscaling:EC2_INSTANCE_LAUNCHING \
    --notification-target-arn arn:aws:sqs:us-west-2:123456789012:my-sqs-queue \
    --role-arn arn:aws:iam::123456789012:role/my-notification-role \
    --notification-metadata "SQS message metadata"
```

This command produces no output.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see PutLifecycleHook in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example adds the specified lifecycle hook to the specified Auto Scaling group.

```
Write-ASLifecycleHook -AutoScalingGroupName my-asg -LifecycleHookName
 "myLifecycleHook" -LifecycleTransition "autoscaling:EC2_INSTANCE_LAUNCHING" -
NotificationTargetARN "arn:aws:sns:us-west-2:123456789012:my-sns-topic" -RoleARN
 "arn:aws:iam::123456789012:role/my-iam-role"
```

- For API details, see PutLifecycleHook in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutNotificationConfiguration` with an AWS SDK or CLI

The following code examples show how to use `PutNotificationConfiguration`.

CLI

AWS CLI

To add a notification

This example adds the specified notification to the specified Auto Scaling group.

```
aws autoscaling put-notification-configuration \
    --auto-scaling-group-name my-asg \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-sns-topic \
    --notification-type autoscaling:TEST_NOTIFICATION
```

This command produces no output.

For more information, see Getting Amazon SNS notifications when your Auto Scaling group scales in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see PutNotificationConfiguration in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example configures the specified Auto Scaling group to send a notification to the specified SNS topic when it launches EC2 instances.

```
Write-ASNotificationConfiguration -AutoScalingGroupName my-asg -
NotificationType "autoscaling:EC2_INSTANCE_LAUNCH" -TopicARN "arn:aws:sns:us-
west-2:123456789012:my-topic"
```

Example 2: This example configures the specified Auto Scaling group to send a notification to the specified SNS topic when it launches or terminates EC2 instances.

```
Write-ASNotificationConfiguration -AutoScalingGroupName my-asg -NotificationType
 @("autoscaling:EC2_INSTANCE_LAUNCH", "autoscaling:EC2_INSTANCE_TERMINATE") -
TopicARN "arn:aws:sns:us-west-2:123456789012:my-topic"
```

- For API details, see PutNotificationConfiguration in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutScalingPolicy` with an AWS SDK or CLI

The following code examples show how to use `PutScalingPolicy`.

CLI

AWS CLI

To add a target tracking scaling policy to an Auto Scaling group

The following `put-scaling-policy` example applies a target tracking scaling policy to the specified Auto Scaling group. The output contains the ARNs and names of the two CloudWatch alarms created on your behalf. If a scaling policy with the same name already exists, it will be overwritten by the new scaling policy.

```
aws autoscaling put-scaling-policy --auto-scaling-group-name my-asg \
  --policy-name alb1000-target-tracking-scaling-policy \
  --policy-type TargetTrackingScaling \
  --target-tracking-configuration file://config.json
```

Contents of `config.json`:

```
{
     "TargetValue": 1000.0,
     "PredefinedMetricSpecification": {
```

```
            "PredefinedMetricType": "ALBRequestCountPerTarget",
            "ResourceLabel": "app/my-alb/778d41231b141a0f/targetgroup/my-alb-
target-group/943f017f100becff"
        }
}
```

Output:

```
{
    "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:228f02c2-
c665-4bfd-aaac-8b04080bea3c:autoScalingGroupName/my-asg:policyName/alb1000-
target-tracking-scaling-policy",
    "Alarms": [
        {
            "AlarmARN": "arn:aws:cloudwatch:region:account-
id:alarm:TargetTracking-my-asg-AlarmHigh-fc0e4183-23ac-497e-9992-691c9980c38e",
            "AlarmName": "TargetTracking-my-asg-AlarmHigh-
fc0e4183-23ac-497e-9992-691c9980c38e"
        },
        {
            "AlarmARN": "arn:aws:cloudwatch:region:account-
id:alarm:TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-bd9e-471a352ee1a2",
            "AlarmName": "TargetTracking-my-asg-AlarmLow-61a39305-ed0c-47af-
bd9e-471a352ee1a2"
        }
    ]
 }
```

For more examples, see Example scaling policies for the AWS Command Line Interface (AWS CLI) in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see PutScalingPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example adds the specified policy to the specified Auto Scaling group. The specified adjustment type determines how to interpret the ScalingAdjustment parameter. With 'ChangeInCapacity', a positive value increases the capacity by the specified number of instances and a negative value decreases the capacity by the specified number of instances.

```
Write-ASScalingPolicy -AutoScalingGroupName my-asg -AdjustmentType
 "ChangeInCapacity" -PolicyName "myScaleInPolicy" -ScalingAdjustment -1
```

Output:

```
arn:aws:autoscaling:us-west-2:123456789012:scalingPolicy:aa3836ab-5462-42c7-adab-
e1d769fc24ef:autoScalingGroupName/my-asg
:policyName/myScaleInPolicy
```

- For API details, see PutScalingPolicy in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutScheduledUpdateGroupAction` with an AWS SDK or CLI

The following code examples show how to use `PutScheduledUpdateGroupAction`.

CLI

AWS CLI

Example 1: To add a scheduled action to an Auto Scaling group

This example adds the specified scheduled action to the specified Auto Scaling group.

```
aws autoscaling put-scheduled-update-group-action \
    --auto-scaling-group-name my-asg \
    --scheduled-action-name my-scheduled-action \
    --start-time "2023-05-12T08:00:00Z" \
    --min-size 2 \
    --max-size 6 \
    --desired-capacity 4
```

This command produces no output. If a scheduled action with the same name already exists, it will be overwritten by the new scheduled action.

For more examples, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To specify a recurring schedule

This example creates a scheduled action to scale on a recurring schedule that is scheduled to execute at 00:30 hours on the first of January, June, and December every year.

```
aws autoscaling put-scheduled-update-group-action \
    --auto-scaling-group-name my-asg \
    --scheduled-action-name my-recurring-action \
    --recurrence "30 0 1 1,6,12 *" \
    --min-size 2 \
    --max-size 6 \
    --desired-capacity 4
```

This command produces no output. If a scheduled action with the same name already exists, it will be overwritten by the new scheduled action.

For more examples, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see PutScheduledUpdateGroupAction in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example creates or updates a one-time scheduled action to change the desired capacity at the specified start time.

```
Write-ASScheduledUpdateGroupAction -AutoScalingGroupName my-asg -
ScheduledActionName "myScheduledAction" -StartTime "2015-12-01T00:00:00Z" -
DesiredCapacity 10
```

- For API details, see PutScheduledUpdateGroupAction in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RecordLifecycleActionHeartbeat` with an AWS SDK or CLI

The following code examples show how to use `RecordLifecycleActionHeartbeat`.

CLI

AWS CLI

To record a lifecycle action heartbeat

This example records a lifecycle action heartbeat to keep the instance in a pending state.

```
aws autoscaling record-lifecycle-action-heartbeat \
    --lifecycle-hook-name my-launch-hook \
    --auto-scaling-group-name my-asg \
    --lifecycle-action-token bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635
```

This command produces no output.

For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see RecordLifecycleActionHeartbeat in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example records a heartbeat for the specified lifecycle action. This keeps the instance in a pending state until you complete the custom action.

```
Write-ASLifecycleActionHeartbeat -AutoScalingGroupName my-asg -LifecycleHookName
 myLifecycleHook -LifecycleActionToken bcd2f1b8-9a78-44d3-8a7a-4dd07d7cf635
```

- For API details, see RecordLifecycleActionHeartbeat in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ResumeProcesses` with an AWS SDK or CLI

The following code examples show how to use `ResumeProcesses`.

CLI

AWS CLI

To resume suspended processes

This example resumes the specified suspended scaling process for the specified Auto Scaling group.

```
aws autoscaling resume-processes \
    --auto-scaling-group-name my-asg \
    --scaling-processes AlarmNotification
```

This command produces no output.

For more information, see Suspending and resuming scaling processes in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see ResumeProcesses in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example resumes the specified Auto Scaling process for the specified Auto Scaling group.

```
Resume-ASProcess -AutoScalingGroupName my-asg -ScalingProcess "AlarmNotification"
```

Example 2: This example resumes all suspended Auto Scaling processes for the specified Auto Scaling group.

```
Resume-ASProcess -AutoScalingGroupName my-asg
```

- For API details, see ResumeProcesses in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetDesiredCapacity` with an AWS SDK or CLI

The following code examples show how to use `SetDesiredCapacity`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
_amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
{desiredCapacity}.");
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see SetDesiredCapacity in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::SetDesiredCapacityRequest request;
        request.SetAutoScalingGroupName(groupName);
        request.SetDesiredCapacity(2);

        Aws::AutoScaling::Model::SetDesiredCapacityOutcome outcome =
                autoScalingClient.SetDesiredCapacity(request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Error with AutoScaling::SetDesiredCapacityRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

        }
```

- For API details, see SetDesiredCapacity in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To set the desired capacity for an Auto Scaling group

This example sets the desired capacity for the specified Auto Scaling group.

```
aws autoscaling set-desired-capacity \
    --auto-scaling-group-name my-asg \
    --desired-capacity 2 \
    --honor-cooldown
```

This command returns to the prompt if successful.

- For API details, see SetDesiredCapacity in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void setDesiredCapacity(AutoScalingClient autoScalingClient,
String groupName) {
       try {
           SetDesiredCapacityRequest capacityRequest =
SetDesiredCapacityRequest.builder()
                   .autoScalingGroupName(groupName)
                   .desiredCapacity(2)
                   .build();

           autoScalingClient.setDesiredCapacity(capacityRequest);
           System.out.println("You have set the DesiredCapacity to 2");

       } catch (AutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
```

- For API details, see SetDesiredCapacity in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun setDesiredCapacity(groupName: String) {
    val capacityRequest =
        SetDesiredCapacityRequest {
            autoScalingGroupName = groupName
            desiredCapacity = 2
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.setDesiredCapacity(capacityRequest)
        println("You set the DesiredCapacity to 2")
    }
}
```

- For API details, see SetDesiredCapacity in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function setDesiredCapacity($autoScalingGroupName, $desiredCapacity)
    {
        return $this->autoScalingClient->setDesiredCapacity([
            'AutoScalingGroupName' => $autoScalingGroupName,
            'DesiredCapacity' => $desiredCapacity,
        ]);
    }
```

- For API details, see SetDesiredCapacity in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example sets the size of the specified Auto Scaling group.

```
Set-ASDesiredCapacity -AutoScalingGroupName my-asg -DesiredCapacity 2
```

Example 2: This example sets the size of the specified Auto Scaling group and waits for the cooldown period to complete before scaling to the new size.

```
Set-ASDesiredCapacity -AutoScalingGroupName my-asg -DesiredCapacity 2 -
HonorCooldown $true
```

- For API details, see SetDesiredCapacity in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def set_desired_capacity(self, group_name: str, capacity: int) -> None:
        """
        Sets the desired capacity of the group. Amazon EC2 Auto Scaling tries to
 keep the
        number of running instances equal to the desired capacity.

        :param group_name: The name of the group to update.
        :param capacity: The desired number of running instances.
        :return: None
        :raises ClientError: If there is an error setting the desired capacity.
        """
        try:
            self.autoscaling_client.set_desired_capacity(
                AutoScalingGroupName=group_name,
                DesiredCapacity=capacity,
                HonorCooldown=False,
            )
            logger.info(
                f"Successfully set desired capacity of {capacity} for Auto
 Scaling group '{group_name}'."
            )
```

```
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(
                f"Failed to set desired capacity for Auto Scaling group
'{group_name}'."
            )
            if error_code == "ScalingActivityInProgress":
                logger.error(
                    f"A scaling activity is currently in progress for the Auto
Scaling group '{group_name}'. "
                    "Please wait for the activity to complete before attempting
to set the desired capacity."
                )
            logger.error(f"Full error:\n\t{err}")
            raise
```

- For API details, see SetDesiredCapacity in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
ScenarioError> {
       // 7. SetDesiredCapacity: set desired capacity to 2.
       //   Wait for a second instance to launch.
       let update_group = self
           .autoscaling
           .set_desired_capacity()
           .auto_scaling_group_name(self.auto_scaling_group_name.clone())
           .desired_capacity(capacity)
           .send()
           .await;
       if let Err(err) = update_group {
```

```
            return Err(ScenarioError::new(
                format!("Failed to update group to desired capacity
({capacity}))").as_str(),
                &err,
            ));
        }
        Ok(())
    }
```

- For API details, see SetDesiredCapacity in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetInstanceHealth` with an AWS SDK or CLI

The following code examples show how to use `SetInstanceHealth`.

CLI

AWS CLI

To set the health status of an instance

This example sets the health status of the specified instance to `Unhealthy`.

```
aws autoscaling set-instance-health \
    --instance-id i-061c63c5eb45f0416 \
    --health-status Unhealthy
```

This command produces no output.

- For API details, see SetInstanceHealth in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example sets the status of the specified instance to 'Unhealthy', taking it out of service. Auto Scaling terminates and replaces the instance.

```
Set-ASInstanceHealth -HealthStatus Unhealthy -InstanceId i-93633f9b
```

Example 2: This example sets the status of the specified instance to 'Healthy', keeping it in service. Any health check grace period for the Auto Scaling group is not honored.

```
Set-ASInstanceHealth -HealthStatus Healthy -InstanceId i-93633f9b -
ShouldRespectGracePeriod $false
```

- For API details, see SetInstanceHealth in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetInstanceProtection` with an AWS SDK or CLI

The following code examples show how to use `SetInstanceProtection`.

CLI

AWS CLI

Example 1: To enable the instance protection setting for an instance

This example enables instance protection for the specified instance.

```
aws autoscaling set-instance-protection \
    --instance-ids i-061c63c5eb45f0416 \
    --auto-scaling-group-name my-asg --protected-from-scale-in
```

This command produces no output.

Example 2: To disable the instance protection setting for an instance

This example disables instance protection for the specified instance.

```
aws autoscaling set-instance-protection \
    --instance-ids i-061c63c5eb45f0416 \
    --auto-scaling-group-name my-asg \
```

```
    --no-protected-from-scale-in
```

This command produces no output.

- For API details, see SetInstanceProtection in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example enables instance protection for the specified instance.

```
Set-ASInstanceProtection -AutoScalingGroupName my-asg -InstanceId i-12345678 -
ProtectedFromScaleIn $true
```

Example 2: This example disables instance protection for the specified instance.

```
Set-ASInstanceProtection -AutoScalingGroupName my-asg -InstanceId i-12345678 -
ProtectedFromScaleIn $false
```

- For API details, see SetInstanceProtection in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SuspendProcesses` with an AWS SDK or CLI

The following code examples show how to use `SuspendProcesses`.

CLI

AWS CLI

To suspend Auto Scaling processes

This example suspends the specified scaling process for the specified Auto Scaling group.

```
aws autoscaling suspend-processes \
```

```
    --auto-scaling-group-name my-asg \
    --scaling-processes AlarmNotification
```

This command produces no output.

For more information, see Suspending and resuming scaling processes in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see SuspendProcesses in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example suspends the specified Auto Scaling process for the specified Auto Scaling group.

```
Suspend-ASProcess -AutoScalingGroupName my-asg -ScalingProcess
 "AlarmNotification"
```

Example 2: This example suspends all Auto Scaling processes for the specified Auto Scaling group.

```
Suspend-ASProcess -AutoScalingGroupName my-asg
```

- For API details, see SuspendProcesses in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `TerminateInstanceInAutoScalingGroup` with an AWS SDK or CLI

The following code examples show how to use `TerminateInstanceInAutoScalingGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics

- Build and manage a resilient service

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::TerminateInstanceInAutoScalingGroupRequest
request;
        request.SetInstanceId(instanceIDs[instanceNumber - 1]);
        request.SetShouldDecrementDesiredCapacity(false);

        Aws::AutoScaling::Model::TerminateInstanceInAutoScalingGroupOutcome
outcome =
                autoScalingClient.TerminateInstanceInAutoScalingGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "Waiting for EC2 instance with ID '"
                      << instanceIDs[instanceNumber - 1] << "' to terminate..."
                      << std::endl;
        }
        else {
            std::cerr << "Error with
AutoScaling::TerminateInstanceInAutoScalingGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
        }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To terminate an instance in an Auto Scaling group

This example terminates the specified instance from the specified Auto Scaling group without updating the size of the group. Amazon EC2 Auto Scaling launches a replacement instance after the specified instance terminates.

```
aws autoscaling terminate-instance-in-auto-scaling-group \
    --instance-id i-061c63c5eb45f0416 \
    --no-should-decrement-desired-capacity
```

Output:

```
{
    "Activities": [
        {
            "ActivityId": "8c35d601-793c-400c-fcd0-f64a27530df7",
            "AutoScalingGroupName": "my-asg",
            "Description": "Terminating EC2 instance: i-061c63c5eb45f0416",
            "Cause": "",
            "StartTime": "2020-10-31T20:34:25.680Z",
            "StatusCode": "InProgress",
            "Progress": 0,
            "Details": "{\"Subnet ID\":\"subnet-6194ea3b\",\"Availability Zone\":
\"us-west-2c\"}"
        }
    ]
}
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void terminateInstanceInAutoScalingGroup(AutoScalingClient
autoScalingClient, String instanceId) {
       try {
           TerminateInstanceInAutoScalingGroupRequest request =
TerminateInstanceInAutoScalingGroupRequest.builder()
                   .instanceId(instanceId)
                   .shouldDecrementDesiredCapacity(false)
                   .build();

           autoScalingClient.terminateInstanceInAutoScalingGroup(request);
           System.out.println("You have terminated instance " + instanceId);

       } catch (AutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun terminateInstanceInAutoScalingGroup(instanceIdVal: String) {
    val request =
        TerminateInstanceInAutoScalingGroupRequest {
            instanceId = instanceIdVal
            shouldDecrementDesiredCapacity = false
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.terminateInstanceInAutoScalingGroup(request)
        println("You have terminated instance $instanceIdVal")
    }
}
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function terminateInstanceInAutoScalingGroup(
        $instanceId,
        $shouldDecrementDesiredCapacity = true,
        $attempts = 0
    ) {
        try {
            return $this->autoScalingClient-
>terminateInstanceInAutoScalingGroup([
                'InstanceId' => $instanceId,
                'ShouldDecrementDesiredCapacity' =>
 $shouldDecrementDesiredCapacity,
            ]);
        } catch (AutoScalingException $exception) {
```

```
            if ($exception->getAwsErrorCode() == "ScalingActivityInProgress" &&
 $attempts < 5) {
                error_log("Cannot terminate an instance while it is still
 pending. Waiting then trying again.");
                sleep(5 * (1 + $attempts));
                return $this->terminateInstanceInAutoScalingGroup(
                    $instanceId,
                    $shouldDecrementDesiredCapacity,
                    ++$attempts
                );
            } else {
                throw $exception;
            }
        }
    }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example terminates the specified instance and decreases the desired capacity of its Auto Scaling group so that Auto Scaling does not launch a replacement instance.

```
Stop-ASInstanceInAutoScalingGroup -InstanceId i-93633f9b -
ShouldDecrementDesiredCapacity $true
```

Output:

```
ActivityId           : 2e40d9bd-1902-444c-abf3-6ea0002efdc5
AutoScalingGroupName :
Cause                : At 2015-11-22T16:09:03Z instance i-93633f9b was taken out
 of service in response to a user
                       request, shrinking the capacity from 2 to 1.
Description          : Terminating EC2 instance: i-93633f9b
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
```

```
EndTime              :
Progress             : 0
StartTime            : 11/22/2015 8:09:03 AM
StatusCode           : InProgress
StatusMessage        :
```

Example 2: This example terminates the specified instance without decreasing the desired capacity of its Auto Scaling group. Auto Scaling launches a replacement instance.

```
Stop-ASInstanceInAutoScalingGroup -InstanceId i-93633f9b -
ShouldDecrementDesiredCapacity $false
```

Output:

```
ActivityId           : 2e40d9bd-1902-444c-abf3-6ea0002efdc5
AutoScalingGroupName :
Cause                : At 2015-11-22T16:09:03Z instance i-93633f9b was taken out
 of service in response to a user
                       request.
Description          : Terminating EC2 instance: i-93633f9b
Details              : {"Availability Zone":"us-west-2b","Subnet
 ID":"subnet-5264e837"}
EndTime              :
Progress             : 0
StartTime            : 11/22/2015 8:09:03 AM
StatusCode           : InProgress
StatusMessage        :
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def terminate_instance(
        self, instance_id: str, decrease_capacity: bool
    ) -> Dict[str, Any]:
        """
        Stops an instance.

        :param instance_id: The ID of the instance to stop.
        :param decrease_capacity: Specifies whether to decrease the desired
 capacity
                                  of the group. When passing True for this
 parameter,
                                  you can stop an instance without having a
 replacement
                                  instance start when the desired capacity
 threshold is
                                  crossed.
        :return: A dictionary containing details of the scaling activity that
 occurs
                 in response to this action.
        :raises ClientError: If there is an error terminating the instance.
        """
        try:
            response =
 self.autoscaling_client.terminate_instance_in_auto_scaling_group(
                InstanceId=instance_id,
 ShouldDecrementDesiredCapacity=decrease_capacity
            )
            logger.info(f"Successfully terminated instance {instance_id}.")
            return response["Activity"]

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(f"Failed to terminate instance {instance_id}.")
```

```
            if error_code == "ScalingActivityInProgress":
                logger.error(
                    "A scaling activity is currently in progress for the Auto
Scaling group "
                    f"associated with instance '{instance_id}'. "
                    "Please wait for the activity to complete before attempting
to terminate the instance."
                )
            elif error_code == "ResourceInUse":
                logger.error(
                    f"The instance '{instance_id}' or an associated resource is
currently in use "
                    "and cannot be terminated. "
                    "Ensure the instance is not involved in any ongoing processes
and try again."
                )
            logger.error(f"Full error:\n\t{err}")
            raise
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    pub async fn terminate_some_instance(&self) -> Result<(), ScenarioError> {
        // Retrieve a list of instances in the auto scaling group.
        let auto_scaling_group = self.get_group().await?;
        let instances = auto_scaling_group.instances();
        // Or use other logic to find an instance to terminate.
        let instance = instances.first();
        if let Some(instance) = instance {
```

```
            let instance_id = if let Some(instance_id) = instance.instance_id() {
                instance_id
            } else {
                return Err(ScenarioError::with("Missing instance id"));
            };
            let termination = self
                .ec2
                .terminate_instances()
                .instance_ids(instance_id)
                .send()
                .await;
            if let Err(err) = termination {
                Err(ScenarioError::new(
                    "There was a problem terminating an instance",
                    &err,
                ))
            } else {
                Ok(())
            }
        } else {
            Err(ScenarioError::with("There was no instance to terminate"))
        }
    }

    async fn get_group(&self) -> Result<AutoScalingGroup, ScenarioError> {
        let describe_auto_scaling_groups = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await;

        if let Err(err) = describe_auto_scaling_groups {
            return Err(ScenarioError::new(
                format!(
                    "Failed to get status of autoscaling group {}",
                    self.auto_scaling_group_name.clone()
                )
                .as_str(),
                &err,
            ));
        }
```

```
        let describe_auto_scaling_groups_output =
describe_auto_scaling_groups.unwrap();
        let auto_scaling_groups =
describe_auto_scaling_groups_output.auto_scaling_groups();
        let auto_scaling_group = auto_scaling_groups.first();

        if auto_scaling_group.is_none() {
            return Err(ScenarioError::with(format!(
                "Could not find autoscaling group {}",
                self.auto_scaling_group_name.clone()
            )));
        }

        Ok(auto_scaling_group.unwrap().clone())
    }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateAutoScalingGroup` with an AWS SDK or CLI

The following code examples show how to use `UpdateAutoScalingGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Build and manage a resilient service

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
```

```
            return true;
        }
        else
        {
            return false;
        }
    }
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::AutoScaling::AutoScalingClient autoScalingClient(clientConfig);

        Aws::AutoScaling::Model::UpdateAutoScalingGroupRequest request;
        request.SetAutoScalingGroupName(groupName);
        request.SetMaxSize(3);

        Aws::AutoScaling::Model::UpdateAutoScalingGroupOutcome outcome =
                autoScalingClient.UpdateAutoScalingGroup(request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Error with AutoScaling::UpdateAutoScalingGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

        }
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To update the size limits of an Auto Scaling group

This example updates the specified Auto Scaling group with a minimum size of 2 and a maximum size of 10.

```
aws autoscaling update-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --min-size 2 \
    --max-size 10
```

This command produces no output.

For more information, see Setting capacity limits for your Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.

Example 2: To add Elastic Load Balancing health checks and specify which Availability Zones and subnets to use

This example updates the specified Auto Scaling group to add Elastic Load Balancing health checks. This command also updates the value of `--vpc-zone-identifier` with a list of subnet IDs in multiple Availability Zones.

```
aws autoscaling update-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --health-check-type ELB \
    --health-check-grace-period 600 \
    --vpc-zone-identifier "subnet-5ea0c127,subnet-6194ea3b,subnet-c934b782"
```

This command produces no output.

For more information, see Elastic Load Balancing and Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Example 3: To update the placement group and termination policy

This example updates the placement group and termination policy to use.

```
aws autoscaling update-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --placement-group my-placement-group \
    --termination-policies "OldestInstance"
```

This command produces no output.

For more information, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.

Example 4: To use the latest version of the launch template

This example updates the specified Auto Scaling group to use the latest version of the specified launch template.

```
aws autoscaling update-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-template LaunchTemplateId=lt-1234567890abcde12,Version='$Latest'
```

This command produces no output.

For more information, see Launch templates in the *Amazon EC2 Auto Scaling User Guide*.

Example 5: To use a specific version of the launch template

This example updates the specified Auto Scaling group to use a specific version of a launch template instead of the latest or default version.

```
aws autoscaling update-auto-scaling-group \
    --auto-scaling-group-name my-asg \
    --launch-template LaunchTemplateName=my-template-for-auto-scaling,Version='2'
```

This command produces no output.

For more information, see Launch templates in the *Amazon EC2 Auto Scaling User Guide*.

Example 6: To define a mixed instances policy and enable capacity rebalancing

This example updates the specified Auto Scaling group to use a mixed instances policy and enables capacity rebalancing. This structure lets you specify groups with Spot and On-Demand capacities and use different launch templates for different architectures.

```
aws autoscaling update-auto-scaling-group \
    --cli-input-json file://~/config.json
```

Contents of `config.json`:

```
{
    "AutoScalingGroupName": "my-asg",
    "CapacityRebalance": true,
    "MixedInstancesPolicy": {
        "LaunchTemplate": {
            "LaunchTemplateSpecification": {
                "LaunchTemplateName": "my-launch-template-for-x86",
                "Version": "$Latest"
            },
            "Overrides": [
                {
                    "InstanceType": "c6g.large",
                    "LaunchTemplateSpecification": {
                        "LaunchTemplateName": "my-launch-template-for-arm",
                        "Version": "$Latest"
                    }
                },
                {
                    "InstanceType": "c5.large"
                },
                {
                    "InstanceType": "c5a.large"
                }
            ]
        },
        "InstancesDistribution": {
            "OnDemandPercentageAboveBaseCapacity": 50,
            "SpotAllocationStrategy": "capacity-optimized"
        }
    }
}
```

This command produces no output.

For more information, see Auto Scaling groups with multiple instance types and purchase options in the *Amazon EC2 Auto Scaling User Guide*.

- For API details, see UpdateAutoScalingGroup in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void updateAutoScalingGroup(AutoScalingClient
autoScalingClient, String groupName,
           String launchTemplateName) {
       try {
           AutoScalingWaiter waiter = autoScalingClient.waiter();
           LaunchTemplateSpecification templateSpecification =
LaunchTemplateSpecification.builder()
                   .launchTemplateName(launchTemplateName)
                   .build();

           UpdateAutoScalingGroupRequest groupRequest =
UpdateAutoScalingGroupRequest.builder()
                   .maxSize(3)
                   .autoScalingGroupName(groupName)
                   .launchTemplate(templateSpecification)
                   .build();

           autoScalingClient.updateAutoScalingGroup(groupRequest);
           DescribeAutoScalingGroupsRequest groupsRequest =
DescribeAutoScalingGroupsRequest.builder()
                   .autoScalingGroupNames(groupName)
                   .build();

           WaiterResponse<DescribeAutoScalingGroupsResponse> waiterResponse =
waiter
                   .waitUntilGroupInService(groupsRequest);
           waiterResponse.matched().response().ifPresent(System.out::println);
```

```
            System.out.println("You successfully updated the auto scaling group
 " + groupName);

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun updateAutoScalingGroup(
    groupName: String,
    launchTemplateNameVal: String,
    serviceLinkedRoleARNVal: String,
) {
    val templateSpecification =
        LaunchTemplateSpecification {
            launchTemplateName = launchTemplateNameVal
        }

    val groupRequest =
        UpdateAutoScalingGroupRequest {
            maxSize = 3
            serviceLinkedRoleArn = serviceLinkedRoleARNVal
            autoScalingGroupName = groupName
            launchTemplate = templateSpecification
        }

    val groupsRequestWaiter =
        DescribeAutoScalingGroupsRequest {
```

```
            autoScalingGroupNames = listOf(groupName)
        }

    AutoScalingClient { region = "us-east-1" }.use { autoScalingClient ->
        autoScalingClient.updateAutoScalingGroup(groupRequest)
        autoScalingClient.waitUntilGroupExists(groupsRequestWaiter)
        println("You successfully updated the Auto Scaling group  $groupName")
    }
}
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public function updateAutoScalingGroup($autoScalingGroupName, $args)
    {
        if (array_key_exists('MaxSize', $args)) {
            $maxSize = ['MaxSize' => $args['MaxSize']];
        } else {
            $maxSize = [];
        }
        if (array_key_exists('MinSize', $args)) {
            $minSize = ['MinSize' => $args['MinSize']];
        } else {
            $minSize = [];
        }
        $parameters = ['AutoScalingGroupName' => $autoScalingGroupName];
        $parameters = array_merge($parameters, $minSize, $maxSize);
        return $this->autoScalingClient->updateAutoScalingGroup($parameters);
    }
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example updates the minimum and maximum size of the specified Auto Scaling group.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -MaxSize 5 -MinSize 1
```

Example 2: This example updates the default cooldown period of the specified Auto Scaling group.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -DefaultCooldown 10
```

Example 3: This example updates the Availability Zones of the specified Auto Scaling group.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -AvailabilityZone @("us-
west-2a", "us-west-2b")
```

Example 4: This example updates the specified Auto Scaling group to use Elastic Load Balancing health checks.

```
Update-ASAutoScalingGroup -AutoScalingGroupName my-asg -HealthCheckType ELB -
HealthCheckGracePeriod 60
```

- For API details, see UpdateAutoScalingGroup in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class AutoScalingWrapper:
    """Encapsulates Amazon EC2 Auto Scaling actions."""

    def __init__(self, autoscaling_client):
        """
        :param autoscaling_client: A Boto3 Amazon EC2 Auto Scaling client.
        """
        self.autoscaling_client = autoscaling_client


    def update_group(self, group_name: str, **kwargs: Any) -> None:
        """
        Updates an Auto Scaling group.

        :param group_name: The name of the group to update.
        :param kwargs: Keyword arguments to pass through to the service.
        :return: None
        :raises ClientError: If there is an error updating the Auto Scaling
 group.
        """
        try:
            self.autoscaling_client.update_auto_scaling_group(
                AutoScalingGroupName=group_name, **kwargs
            )
            logger.info(f"Successfully updated Auto Scaling group {group_name}.")

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            logger.error(f"Failed to update Auto Scaling group {group_name}.")
            if error_code == "ResourceInUse":
                logger.error(
```

```
                    "The Auto Scaling group '%s' is currently in use and cannot
 be modified. Please try again later.",
                    group_name,
                )
            elif error_code == "ScalingActivityInProgress":
                logger.error(
                    f"A scaling activity is currently in progress for the Auto
 Scaling group '{group_name}'."
                    "Please wait for the activity to complete before attempting
 to update the group."
                )
            logger.error(f"Full error:\n\t{err}")
            raise
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn update_group(client: &Client, name: &str, size: i32) -> Result<(),
 Error> {
    client
        .update_auto_scaling_group()
        .auto_scaling_group_name(name)
        .max_size(size)
        .send()
        .await?;

    println!("Updated AutoScaling group");

    Ok(())
}
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Auto Scaling using AWS SDKs

The following code examples show you how to implement common scenarios in Auto Scaling with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Auto Scaling or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Build and manage a resilient service using an AWS SDK

Build and manage a resilient service using an AWS SDK

The following code examples show how to create a load-balanced web service that returns book, movie, and song recommendations. The example shows how the service responds to failures, and how to restructure the service for more resilience when failures occur.

- Use an Amazon EC2 Auto Scaling group to create Amazon Elastic Compute Cloud (Amazon EC2) instances based on a launch template and to keep the number of instances in a specified range.
- Handle and distribute HTTP requests with Elastic Load Balancing.
- Monitor the health of instances in an Auto Scaling group and forward requests only to healthy instances.
- Run a Python web server on each EC2 instance to handle HTTP requests. The web server responds with recommendations and health checks.
- Simulate a recommendation service with an Amazon DynamoDB table.

- Control web server response to requests and health checks by updating AWS Systems Manager parameters.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
```

```
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
```

```
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper =
host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several
AWS resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways
to make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
```

```
            "\t* A DynamoDB table that the web service depends on to provide
book, movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each
contain a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across
several Availability Zones.");
        Console.WriteLine(
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the
Auto Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs
'server_startup_script.sh' when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the
`server.py` script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to
'/' and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
```

```
            + "permissions to access the DynamoDB recommendation table and
Systems Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2
instances, each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3,
_autoScalerWrapper.GroupName, zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or
continue with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the
demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");
```

```
        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
 _autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
 _elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGro
 protocol, port, defaultVpc.VpcId);

        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.Lo
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port,
 ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for
 your default VPC must\n"
```

```
                    + "allows access from this computer. You can either add it
automatically from this\n"
                    + "example or add it yourself using the AWS Management
Console.\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
inbound traffic from your computer's IP address?"))
                {
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port,
ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
                }
            }
            loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
```

```
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
                          "to create situations where the web service fails, and
shows how using a resilient\n" +
                          "architecture can keep the web service running in spite
of these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
```

```
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
"this-is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service
fails, the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in
the target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
```

```
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile
that contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns
either a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo,
a deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not
for Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the
Auto Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load
balancer can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two
for the load balancer to detect the unhealthy");
```

```
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by
an auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler
start a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a
successful recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the
load balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance
typically takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
"this-is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
```

```
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account,
we can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBal
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGr
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupNa
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
```

```
                "Don't forget to delete them when you're done with them or you
 might incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Create a class that wraps Auto Scaling and Amazon EC2 actions.

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;
    private readonly ILogger<AutoScalerWrapper> _logger;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;
```

```
    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration,
        ILogger<AutoScalerWrapper> logger)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;
        _logger = logger;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with
 a specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
 granted to
    /// clients that run on the instance.
    /// </summary>
```

```
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to
 the role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                    "\"Version\": \"2012-10-17\"," +
                                    "\"Statement\": [{" +
                                         "\"Effect\": \"Allow\"," +
                                         "\"Principal\": {" +
                                         "\"Service\": [" +
                                             "\"ec2.amazonaws.com\"" +
                                         "]" +
                                         "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
```

```
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
```

```
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await
 _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto
Scaling.
    /// The launch template specifies a Bash script in its user data field that
runs after
    /// the instance is started. This script installs the Python packages and
starts a Python
    /// web server on the instance.
```

```
   /// </summary>
   /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
   /// <param name="instancePolicyPath">The path to a permissions policy to
create and attach to the profile.</param>
   /// <returns>The template object.</returns>
   public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
   {
       try
       {
           await CreateKeyPair(_keyPairName);
           await CreateInstanceProfileWithName(_instancePolicyName,
_instanceRoleName,
               _instanceProfileName, instancePolicyPath);

           var startServerText = await File.ReadAllTextAsync(startupScriptPath);
           var plainTextBytes =
System.Text.Encoding.UTF8.GetBytes(startServerText);

           var amiLatest = await _amazonSsm.GetParameterAsync(
               new GetParameterRequest() { Name = _amiParam });
           var amiId = amiLatest.Parameter.Value;
           var launchTemplateResponse = await
_amazonEc2.CreateLaunchTemplateAsync(
               new CreateLaunchTemplateRequest()
               {
                   LaunchTemplateName = _launchTemplateName,
                   LaunchTemplateData = new RequestLaunchTemplateData()
                   {
                       InstanceType = _instanceType,
                       ImageId = amiId,
                       IamInstanceProfile =
                           new

LaunchTemplateIamInstanceProfileSpecificationRequest()
                           {
                               Name = _instanceProfileName
                           },
                       KeyName = _keyPairName,
                       UserData = System.Convert.ToBase64String(plainTextBytes)
                   }
               });
           return launchTemplateResponse.LaunchTemplate;
```

```
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.AlreadyExistsException")
            {
                _logger.LogError($"Could not create the template, the name
{_launchTemplateName} already exists. " +
                                $"Please try again with a unique name.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2
Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        try
        {
            var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
                new DescribeAvailabilityZonesRequest());
            return zoneResponse.AvailabilityZones.Select(z =>
z.ZoneName).ToList();
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            _logger.LogError($"An Amazon EC2 error occurred while listing
availability zones.: {ec2Exception.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"An error occurred while listing availability
 zones.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
 List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new
 Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with
 size {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
 exists.");
        }
    }
```

```
    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        try
        {
            var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
                new DescribeVpcsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("is-default", new List<string>() { "true" })
                    }
                });
            return vpcResponse.Vpcs[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "UnauthorizedOperation")
            {
                _logger.LogError(ec2Exception, $"You do not have the necessary
 permissions to describe VPCs.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while describing the vpcs.:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
 List<string> availabilityZones)
```

```
    {
        try
        {
            var subnets = new List<Subnet>();
            var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
                new DescribeSubnetsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("vpc-id", new List<string>() { vpcId }),
                        new("availability-zone", availabilityZones),
                        new("default-for-az", new List<string>() { "true" })
                    }
                });

            // Get the entire list using the paginator.
            await foreach (var subnet in subnetPaginator.Subnets)
            {
                subnets.Add(subnet);
            }

            return subnets;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
            {
                _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId}
does not exist.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while describing the
subnets.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
```

```
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.NotFoundException")
            {
                _logger.LogError(
                    $"Could not delete the template, the name
{_launchTemplateName} was not found.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the
role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
```

```
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await
_amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
```

```
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        try
        {
            var response = await
_amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
                new DescribeIamInstanceProfileAssociationsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("instance-id", new List<string>() { instanceId })
                    },
                });
            return response.IamInstanceProfileAssociations[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
            {
                _logger.LogError(ec2Exception, $"Instance {instanceId} not
found");
            }
```

```
            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while creating the
template.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile
is replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate
with the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association
for the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        try
        {
            await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
            // Allow time before resetting.
            Thread.Sleep(25000);

            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(25000);
            var instanceReady = false;
```

```
            var retries = 5;
            while (retries-- > 0 && !instanceReady)
            {
                var instancesPaginator =
                    _amazonSsm.Paginators.DescribeInstanceInformation(
                        new DescribeInstanceInformationRequest());
                // Get the entire list using the paginator.
                await foreach (var instance in
instancesPaginator.InstanceInformationList)
                {
                    instanceReady = instance.InstanceId == instanceId;
                    if (instanceReady)
                    {
                        break;
                    }
                }
            }
            Console.WriteLine("Waiting for instance to be running.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);
            Console.WriteLine("Instance ready.");
            Console.WriteLine($"Sending restart command to instance
{instanceId}");
            await _amazonSsm.SendCommandAsync(
                new SendCommandRequest()
                {
                    InstanceIds = new List<string>() { instanceId },
                    DocumentName = "AWS-RunShellScript",
                    Parameters = new Dictionary<string, List<string>>()
                    {
                        {
                            "commands",
                            new List<string>() { "cd / && sudo python3 server.py
80" }
                        }
                    }
                });
            Console.WriteLine($"Restarted the web server on instance
{instanceId}");
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
            {
```

```
                _logger.LogError(ec2Exception, $"Instance {instanceId} not
found");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while replacing the
template.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await
_amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for
{instanceId}. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }
```

```
    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string
groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
```

```
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
```

```
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the
calling computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP
address while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }

                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be
open to either this computer's IP\n" +
                                      "address, to all IP addresses (0.0.0.0/0),
or to a prefix list ID.");
```

```
                }
                else
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
 the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
```

```
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName,
 string targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
 InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is in the specified state.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEc2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
```

```
            Console.Write(". ");
        } while (!hasState);

        return hasState;
    }
}
```

Create a class that wraps Elastic Load Balancing actions.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
```

```
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
```

```
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
 groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened
 times and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</
param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer
 exists.</param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
```

```
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
 subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
 List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
 _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
```

```
                    await
_amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
```

```
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await
_amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn =
targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
```

```
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

Create a class that uses DynamoDB to simulate a recommendation service.

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books,
 movies, and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
```

```
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
                            AttributeName = "ItemId",
                            AttributeType = ScalarAttributeType.N
                        }
                    },
                KeySchema = new List<KeySchemaElement>()
                    {
                        new KeySchemaElement()
                        {
                            AttributeName = "MediaType",
                            KeyType = KeyType.HASH
                        },
                        new KeySchemaElement()
                        {
                            AttributeName = "ItemId",
                            KeyType = KeyType.RANGE
                        }
                    },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 5,
```

```
                    WriteCapacityUnits = 5
                }
            };
            await _amazonDynamoDb.CreateTableAsync(createRequest);

            // Wait until the table is ACTIVE and then report success.
            Console.Write("\nWaiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            TableStatus status;
            do
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
 _amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
```

```
    {
        var recommendationsText = await
 File.ReadAllTextAsync(recommendationsPath);
        var records =

 JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Create a class that wraps Systems Manager actions.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
```

```
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement
 _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-
architecture-table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
```

```
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 true });
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets

- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
public class Main {

    public static final String fileName = "C:\\AWS\\resworkflow\
\recommendations.json"; // Modify file location.
    public static final String tableName = "doc-example-recommendation-service";
    public static final String startScript = "C:\\AWS\\resworkflow\
\server_startup_script.sh"; // Modify file location.
    public static final String policyFile = "C:\\AWS\\resworkflow\
\instance_policy.json"; // Modify file location.
    public static final String ssmJSON = "C:\\AWS\\resworkflow\
\ssm_only_policy.json"; // Modify file location.
    public static final String failureResponse = "doc-example-resilient-
architecture-failure-response";
    public static final String healthCheck = "doc-example-resilient-architecture-
health-check";
    public static final String templateName = "doc-example-resilience-template";
    public static final String roleName = "doc-example-resilience-role";
    public static final String policyName = "doc-example-resilience-pol";
    public static final String profileName = "doc-example-resilience-prof";
```

```
    public static final String badCredsProfileName = "doc-example-resilience-
prof-bc";

    public static final String targetGroupName = "doc-example-resilience-tg";
    public static final String autoScalingGroupName = "doc-example-resilience-
group";
    public static final String lbName = "doc-example-resilience-lb";
    public static final String protocol = "HTTP";
    public static final int port = 80;

    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException,
 InterruptedException {
        Scanner in = new Scanner(System.in);
        Database database = new Database();
        AutoScaler autoScaler = new AutoScaler();
        LoadBalancer loadBalancer = new LoadBalancer();

        System.out.println(DASHES);
        System.out.println("Welcome to the demonstration of How to Build and
 Manage a Resilient Service!");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("A - SETUP THE RESOURCES");
        System.out.println("Press Enter when you're ready to start deploying
 resources.");
        in.nextLine();
        deploy(loadBalancer);
        System.out.println(DASHES);
        System.out.println(DASHES);
        System.out.println("B - DEMO THE RESILIENCE FUNCTIONALITY");
        System.out.println("Press Enter when you're ready.");
        in.nextLine();
        demo(loadBalancer);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("C - DELETE THE RESOURCES");
        System.out.println("""
```

```
                This concludes the demo of how to build and manage a resilient
service.
                To keep things tidy and to avoid unwanted charges on your
account, we can clean up all AWS resources
                that were created for this demo.
                """);

        System.out.println("\n Do you want to delete the resources (y/n)? ");
        String userInput = in.nextLine().trim().toLowerCase(); // Capture user
input

        if (userInput.equals("y")) {
            // Delete resources here
            deleteResources(loadBalancer, autoScaler, database);
            System.out.println("Resources deleted.");
        } else {
            System.out.println("""
                    Okay, we'll leave the resources intact.
                    Don't forget to delete them when you're done with them or you
might incur unexpected charges.
                    """);
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The example has completed. ");
        System.out.println("\n Thanks for watching!");
        System.out.println(DASHES);
    }

    // Deletes the AWS resources used in this example.
    private static void deleteResources(LoadBalancer loadBalancer, AutoScaler
autoScaler, Database database)
            throws IOException, InterruptedException {
        loadBalancer.deleteLoadBalancer(lbName);
        System.out.println("*** Wait 30 secs for resource to be deleted");
        TimeUnit.SECONDS.sleep(30);
        loadBalancer.deleteTargetGroup(targetGroupName);
        autoScaler.deleteAutoScaleGroup(autoScalingGroupName);
        autoScaler.deleteRolesPolicies(policyName, roleName, profileName);
        autoScaler.deleteTemplate(templateName);
        database.deleteTable(tableName);
    }
```

```
   private static void deploy(LoadBalancer loadBalancer) throws
InterruptedException, IOException {
       Scanner in = new Scanner(System.in);
       System.out.println(
               """
                       For this demo, we'll use the AWS SDK for Java (v2) to
create several AWS resources
                       to set up a load-balanced web service endpoint and
explore some ways to make it resilient
                       against various kinds of failures.

                       Some of the resources create by this demo are:
                       \t* A DynamoDB table that the web service depends on to
provide book, movie, and song recommendations.
                       \t* An EC2 launch template that defines EC2 instances
that each contain a Python web server.
                       \t* An EC2 Auto Scaling group that manages EC2 instances
across several Availability Zones.
                       \t* An Elastic Load Balancing (ELB) load balancer that
targets the Auto Scaling group to distribute requests.
                       """);

       System.out.println("Press Enter when you're ready.");
       in.nextLine();
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("Creating and populating a DynamoDB table named " +
tableName);
       Database database = new Database();
       database.createTable(tableName, fileName);
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("""
               Creating an EC2 launch template that runs '{startup_script}' when
an instance starts.
               This script starts a Python web server defined in the `server.py`
script. The web server
               listens to HTTP requests on port 80 and responds to requests to
'/' and to '/healthcheck'.
               For demo purposes, this server is run as the root user. In
production, the best practice is to
```

```
                run a web server, such as Apache, with least-privileged
credentials.

                The template also defines an IAM policy that each instance uses
to assume a role that grants
                permissions to access the DynamoDB recommendation table and
Systems Manager parameters
                that control the flow of the demo.
                """);

        LaunchTemplateCreator templateCreator = new LaunchTemplateCreator();
        templateCreator.createTemplate(policyFile, policyName, profileName,
startScript, templateName, roleName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
                "Creating an EC2 Auto Scaling group that maintains three EC2
instances, each in a different Availability Zone.");
        System.out.println("*** Wait 30 secs for the VPC to be created");
        TimeUnit.SECONDS.sleep(30);
        AutoScaler autoScaler = new AutoScaler();
        String[] zones = autoScaler.createGroup(3, templateName,
autoScalingGroupName);

        System.out.println("""
                At this point, you have EC2 instances created. Once each instance
starts, it listens for
                HTTP requests. You can see these instances in the console or
continue with the demo.
                Press Enter when you're ready to continue.
                """);

        in.nextLine();
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Creating variables that control the flow of the
demo.");
        ParameterHelper paramHelper = new ParameterHelper();
        paramHelper.reset();
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("""
                Creating an Elastic Load Balancing target group and load
balancer. The target group
                defines how the load balancer connects to instances. The load
balancer provides a
                single endpoint where clients connect and dispatches requests to
instances in the group.
                """);

        String vpcId = autoScaler.getDefaultVPC();
        List<Subnet> subnets = autoScaler.getSubnets(vpcId, zones);
        System.out.println("You have retrieved a list with " + subnets.size() + "
subnets");
        String targetGroupArn = loadBalancer.createTargetGroup(protocol, port,
vpcId, targetGroupName);
        String elbDnsName = loadBalancer.createLoadBalancer(subnets,
targetGroupArn, lbName, port, protocol);
        autoScaler.attachLoadBalancerTargetGroup(autoScalingGroupName,
targetGroupArn);
        System.out.println("Verifying access to the load balancer endpoint...");
        boolean wasSuccessul =
loadBalancer.verifyLoadBalancerEndpoint(elbDnsName);
        if (!wasSuccessul) {
            System.out.println("Couldn't connect to the load balancer, verifying
that the port is open...");
            CloseableHttpClient httpClient = HttpClients.createDefault();

            // Create an HTTP GET request to "http://checkip.amazonaws.com"
            HttpGet httpGet = new HttpGet("http://checkip.amazonaws.com");
            try {
                // Execute the request and get the response
                HttpResponse response = httpClient.execute(httpGet);

                // Read the response content.
                String ipAddress =
IOUtils.toString(response.getEntity().getContent(),
StandardCharsets.UTF_8).trim();

                // Print the public IP address.
                System.out.println("Public IP Address: " + ipAddress);
                GroupInfo groupInfo = autoScaler.verifyInboundPort(vpcId, port,
ipAddress);
                if (!groupInfo.isPortOpen()) {
                    System.out.println("""
```

```
                            For this example to work, the default security group
for your default VPC must
                            allow access from this computer. You can either add
it automatically from this
                            example or add it yourself using the AWS Management
Console.
                            """);

                    System.out.println(
                            "Do you want to add a rule to security group " +
groupInfo.getGroupName() + " to allow");
                    System.out.println("inbound traffic on port " + port + " from
your computer's IP address (y/n) ");
                    String ans = in.nextLine();
                    if ("y".equalsIgnoreCase(ans)) {
                        autoScaler.openInboundPort(groupInfo.getGroupName(),
String.valueOf(port), ipAddress);
                        System.out.println("Security group rule added.");
                    } else {
                        System.out.println("No security group rule added.");
                    }
                }

            } catch (AutoScalingException e) {
                e.printStackTrace();
            }
        } else if (wasSuccessul) {
            System.out.println("Your load balancer is ready. You can access it by
browsing to:");
            System.out.println("\t http://" + elbDnsName);
        } else {
            System.out.println("Couldn't get a successful response from the load
balancer endpoint. Troubleshoot by");
            System.out.println("manually verifying that your VPC and security
group are configured correctly and that");
            System.out.println("you can successfully make a GET request to the
load balancer.");
        }

        System.out.println("Press Enter when you're ready to continue with the
demo.");
        in.nextLine();
    }
```

```
   // A method that controls the demo part of the Java program.
   public static void demo(LoadBalancer loadBalancer) throws IOException,
InterruptedException {
       ParameterHelper paramHelper = new ParameterHelper();
       System.out.println("Read the ssm_only_policy.json file");
       String ssmOnlyPolicy = readFileAsString(ssmJSON);

       System.out.println("Resetting parameters to starting values for demo.");
       paramHelper.reset();

       System.out.println(
               """
                       This part of the demonstration shows how to toggle
different parts of the system
                       to create situations where the web service fails, and
shows how using a resilient
                       architecture can keep the web service running in spite
of these failures.

                       At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.
                      """);
       demoChoices(loadBalancer);

       System.out.println(
               """
                       The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.
                       The table name is contained in a Systems Manager
parameter named self.param_helper.table.
                       To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.
                      """);
       paramHelper.put(paramHelper.tableName, "this-is-not-a-table");

       System.out.println(
               """
                       \nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as
                       healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.
                      """);
       demoChoices(loadBalancer);
```

```
        System.out.println(
                """
                        Instead of failing when the recommendation service fails,
the web service can return a static response.
                        While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.
                        """);
        paramHelper.put(paramHelper.failureResponse, "static");

        System.out.println("""
                Now, sending a GET request to the load balancer endpoint returns
a static response.
                The service still reports as healthy because health checks are
still shallow.
                """);
        demoChoices(loadBalancer);

        System.out.println("Let's reinstate the recommendation service.");
        paramHelper.put(paramHelper.tableName, paramHelper.dyntable);

        System.out.println("""
                Let's also substitute bad credentials for one of the instances in
the target group so that it can't
                access the DynamoDB recommendation table. We will get an instance
id value.
                """);

        LaunchTemplateCreator templateCreator = new LaunchTemplateCreator();
        AutoScaler autoScaler = new AutoScaler();

        // Create a new instance profile based on badCredsProfileName.
        templateCreator.createInstanceProfile(policyFile, policyName,
badCredsProfileName, roleName);
        String badInstanceId = autoScaler.getBadInstance(autoScalingGroupName);
        System.out.println("The bad instance id values used for this demo is " +
badInstanceId);

        String profileAssociationId =
autoScaler.getInstanceProfile(badInstanceId);
        System.out.println("The association Id value is " +
profileAssociationId);
        System.out.println("Replacing the profile for instance " + badInstanceId
                + " with a profile that contains bad credentials");
```

```
        autoScaler.replaceInstanceProfile(badInstanceId, badCredsProfileName,
profileAssociationId);

        System.out.println(
                """
                        Now, sending a GET request to the load balancer endpoint
returns either a recommendation or a static response,
                        depending on which instance is selected by the load
balancer.
                        """);

        demoChoices(loadBalancer);

        System.out.println("""
                Let's implement a deep health check. For this demo, a deep health
check tests whether
                the web service can access the DynamoDB table that it depends on
for recommendations. Note that
                the deep health check is only for ELB routing and not for Auto
Scaling instance health.
                This kind of deep health check is not recommended for Auto
Scaling instance health, because it
                risks accidental termination of all instances in the Auto Scaling
group when a dependent service fails.
                """);

        System.out.println("""
                By implementing deep health checks, the load balancer can detect
when one of the instances is failing
                and take that instance out of rotation.
                """);

        paramHelper.put(paramHelper.healthCheck, "deep");

        System.out.println("""
                Now, checking target health indicates that the instance with bad
credentials
                is unhealthy. Note that it might take a minute or two for the
load balancer to detect the unhealthy
                instance. Sending a GET request to the load balancer endpoint
always returns a recommendation, because
                the load balancer takes unhealthy instances out of its rotation.
                """);
```

```
        demoChoices(loadBalancer);

        System.out.println(
                """
                        Because the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy
                        instance is to terminate it and let the auto scaler start
a new instance to replace it.
                        """);
        autoScaler.terminateInstance(badInstanceId);

        System.out.println("""
                Even while the instance is terminating and the new instance is
starting, sending a GET
                request to the web service continues to get a successful
recommendation response because
                the load balancer routes requests to the healthy instances. After
the replacement instance
                starts and reports as healthy, it is included in the load
balancing rotation.
                Note that terminating and replacing an instance typically takes
several minutes, during which time you
                can see the changing health check status until the new instance
is running and healthy.
                """);

        demoChoices(loadBalancer);
        System.out.println(
                "If the recommendation service fails now, deep health checks mean
all instances report as unhealthy.");
        paramHelper.put(paramHelper.tableName, "this-is-not-a-table");

        demoChoices(loadBalancer);
        paramHelper.reset();
    }

    public static void demoChoices(LoadBalancer loadBalancer) throws IOException,
InterruptedException {
        String[] actions = {
                "Send a GET request to the load balancer endpoint.",
                "Check the health of load balancer targets.",
                "Go to the next part of the demo."
        };
        Scanner scanner = new Scanner(System.in);
```

```
        while (true) {
            System.out.println("-".repeat(88));
            System.out.println("See the current state of the service by selecting
one of the following choices:");
            for (int i = 0; i < actions.length; i++) {
                System.out.println(i + ": " + actions[i]);
            }

            try {
                System.out.print("\nWhich action would you like to take? ");
                int choice = scanner.nextInt();
                System.out.println("-".repeat(88));

                switch (choice) {
                    case 0 -> {
                        System.out.println("Request:\n");
                        System.out.println("GET http://" +
loadBalancer.getEndpoint(lbName));
                        CloseableHttpClient httpClient =
HttpClients.createDefault();

                        // Create an HTTP GET request to the ELB.
                        HttpGet httpGet = new HttpGet("http://" +
loadBalancer.getEndpoint(lbName));

                        // Execute the request and get the response.
                        HttpResponse response = httpClient.execute(httpGet);
                        int statusCode =
response.getStatusLine().getStatusCode();
                        System.out.println("HTTP Status Code: " + statusCode);

                        // Display the JSON response
                        BufferedReader reader = new BufferedReader(
                                new
InputStreamReader(response.getEntity().getContent()));
                        StringBuilder jsonResponse = new StringBuilder();
                        String line;
                        while ((line = reader.readLine()) != null) {
                            jsonResponse.append(line);
                        }
                        reader.close();

                        // Print the formatted JSON response.
```

```
                        System.out.println("Full Response:\n");
                        System.out.println(jsonResponse.toString());

                        // Close the HTTP client.
                        httpClient.close();

                    }
                    case 1 -> {
                        System.out.println("\nChecking the health of load
balancer targets:\n");
                        List<TargetHealthDescription> health =
loadBalancer.checkTargetHealth(targetGroupName);
                        for (TargetHealthDescription target : health) {
                            System.out.printf("\tTarget %s on port %d is %s%n",
target.target().id(),
                                    target.target().port(),
target.targetHealth().stateAsString());
                        }
                        System.out.println("""
                                Note that it can take a minute or two for the
health check to update
                                after changes are made.
                                """);
                    }
                    case 2 -> {
                        System.out.println("\nOkay, let's move on.");
                        System.out.println("-".repeat(88));
                        return; // Exit the method when choice is 2
                    }
                    default -> System.out.println("You must choose a value
between 0-2. Please select again.");
                }

            } catch (java.util.InputMismatchException e) {
                System.out.println("Invalid input. Please select again.");
                scanner.nextLine(); // Clear the input buffer.
            }
        }
    }

    public static String readFileAsString(String filePath) throws IOException {
        byte[] bytes = Files.readAllBytes(Paths.get(filePath));
        return new String(bytes);
    }
```

```
}
```

Create a class that wraps Auto Scaling and Amazon EC2 actions.

```
public class AutoScaler {

    private static Ec2Client ec2Client;
    private static AutoScalingClient autoScalingClient;
    private static IamClient iamClient;

    private static SsmClient ssmClient;

    private IamClient getIAMClient() {
        if (iamClient == null) {
            iamClient = IamClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return iamClient;
    }

    private SsmClient getSSMClient() {
        if (ssmClient == null) {
            ssmClient = SsmClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return ssmClient;
    }

    private Ec2Client getEc2Client() {
        if (ec2Client == null) {
            ec2Client = Ec2Client.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return ec2Client;
    }

    private AutoScalingClient getAutoScalingClient() {
        if (autoScalingClient == null) {
            autoScalingClient = AutoScalingClient.builder()
```

```
                    .region(Region.US_EAST_1)
                    .build();
        }
        return autoScalingClient;
    }

    /**
     * Terminates and instances in an EC2 Auto Scaling group. After an instance
is
     * terminated, it can no longer be accessed.
     */
    public void terminateInstance(String instanceId) {
        TerminateInstanceInAutoScalingGroupRequest terminateInstanceIRequest =
TerminateInstanceInAutoScalingGroupRequest
                .builder()
                .instanceId(instanceId)
                .shouldDecrementDesiredCapacity(false)
                .build();


getAutoScalingClient().terminateInstanceInAutoScalingGroup(terminateInstanceIRequest);
        System.out.format("Terminated instance %s.", instanceId);
    }

    /**
     * Replaces the profile associated with a running instance. After the profile
is
     * replaced, the instance is rebooted to ensure that it uses the new profile.
     * When
     * the instance is ready, Systems Manager is used to restart the Python web
     * server.
     */
    public void replaceInstanceProfile(String instanceId, String
newInstanceProfileName, String profileAssociationId)
            throws InterruptedException {
        // Create an IAM instance profile specification.
        software.amazon.awssdk.services.ec2.model.IamInstanceProfileSpecification
iamInstanceProfile =
software.amazon.awssdk.services.ec2.model.IamInstanceProfileSpecification
                .builder()
                .name(newInstanceProfileName) // Make sure
'newInstanceProfileName' is a valid IAM Instance Profile
                                                // name.
                .build();
```

```
        // Replace the IAM instance profile association for the EC2 instance.
        ReplaceIamInstanceProfileAssociationRequest replaceRequest =
ReplaceIamInstanceProfileAssociationRequest
                .builder()
                .iamInstanceProfile(iamInstanceProfile)
                .associationId(profileAssociationId) // Make sure
'profileAssociationId' is a valid association ID.
                .build();

        try {
            getEc2Client().replaceIamInstanceProfileAssociation(replaceRequest);
            // Handle the response as needed.
        } catch (Ec2Exception e) {
            // Handle exceptions, log, or report the error.
            System.err.println("Error: " + e.getMessage());
        }
        System.out.format("Replaced instance profile for association %s with
profile %s.", profileAssociationId,
                newInstanceProfileName);
        TimeUnit.SECONDS.sleep(15);
        boolean instReady = false;
        int tries = 0;

        // Reboot after 60 seconds
        while (!instReady) {
            if (tries % 6 == 0) {
                getEc2Client().rebootInstances(RebootInstancesRequest.builder()
                        .instanceIds(instanceId)
                        .build());
                System.out.println("Rebooting instance " + instanceId + " and
waiting for it to be ready.");
            }
            tries++;
            try {
                TimeUnit.SECONDS.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            DescribeInstanceInformationResponse informationResponse =
getSSMClient().describeInstanceInformation();
            List<InstanceInformation> instanceInformationList =
informationResponse.instanceInformationList();
```

```
            for (InstanceInformation info : instanceInformationList) {
                if (info.instanceId().equals(instanceId)) {
                    instReady = true;
                    break;
                }
            }
        }

        SendCommandRequest sendCommandRequest = SendCommandRequest.builder()
                .instanceIds(instanceId)
                .documentName("AWS-RunShellScript")
                .parameters(Collections.singletonMap("commands",
                        Collections.singletonList("cd / && sudo python3 server.py
80")))
                .build();

        getSSMClient().sendCommand(sendCommandRequest);
        System.out.println("Restarted the Python web server on instance " +
instanceId + ".");
    }

    public void openInboundPort(String secGroupId, String port, String ipAddress)
{
        AuthorizeSecurityGroupIngressRequest ingressRequest =
AuthorizeSecurityGroupIngressRequest.builder()
                .groupName(secGroupId)
                .cidrIp(ipAddress)
                .fromPort(Integer.parseInt(port))
                .build();

        getEc2Client().authorizeSecurityGroupIngress(ingressRequest);
        System.out.format("Authorized ingress to %s on port %s from %s.",
secGroupId, port, ipAddress);
    }

    /**
     * Detaches a role from an instance profile, detaches policies from the role,
     * and deletes all the resources.
     */
    public void deleteInstanceProfile(String roleName, String profileName) {
        try {
            software.amazon.awssdk.services.iam.model.GetInstanceProfileRequest
getInstanceProfileRequest =
software.amazon.awssdk.services.iam.model.GetInstanceProfileRequest
```

```
                    .builder()
                    .instanceProfileName(profileName)
                    .build();

            GetInstanceProfileResponse response =
getIAMClient().getInstanceProfile(getInstanceProfileRequest);
            String name = response.instanceProfile().instanceProfileName();
            System.out.println(name);

            RemoveRoleFromInstanceProfileRequest profileRequest =
RemoveRoleFromInstanceProfileRequest.builder()
                    .instanceProfileName(profileName)
                    .roleName(roleName)
                    .build();

            getIAMClient().removeRoleFromInstanceProfile(profileRequest);
            DeleteInstanceProfileRequest deleteInstanceProfileRequest =
DeleteInstanceProfileRequest.builder()
                    .instanceProfileName(profileName)
                    .build();

            getIAMClient().deleteInstanceProfile(deleteInstanceProfileRequest);
            System.out.println("Deleted instance profile " + profileName);

            DeleteRoleRequest deleteRoleRequest = DeleteRoleRequest.builder()
                    .roleName(roleName)
                    .build();

            // List attached role policies.
            ListAttachedRolePoliciesResponse rolesResponse = getIAMClient()
                    .listAttachedRolePolicies(role -> role.roleName(roleName));
            List<AttachedPolicy> attachedPolicies =
rolesResponse.attachedPolicies();
            for (AttachedPolicy attachedPolicy : attachedPolicies) {
                DetachRolePolicyRequest request =
DetachRolePolicyRequest.builder()
                        .roleName(roleName)
                        .policyArn(attachedPolicy.policyArn())
                        .build();

                getIAMClient().detachRolePolicy(request);
                System.out.println("Detached and deleted policy " +
attachedPolicy.policyName());
            }
```

```
            getIAMClient().deleteRole(deleteRoleRequest);
            System.out.println("Instance profile and role deleted.");

        } catch (IamException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public void deleteTemplate(String templateName) {
        getEc2Client().deleteLaunchTemplate(name ->
name.launchTemplateName(templateName));
        System.out.format(templateName + " was deleted.");
    }

    public void deleteAutoScaleGroup(String groupName) {
        DeleteAutoScalingGroupRequest deleteAutoScalingGroupRequest =
DeleteAutoScalingGroupRequest.builder()
                .autoScalingGroupName(groupName)
                .forceDelete(true)
                .build();


getAutoScalingClient().deleteAutoScalingGroup(deleteAutoScalingGroupRequest);
        System.out.println(groupName + " was deleted.");
    }

    /*
     * Verify the default security group of the specified VPC allows ingress from
     * this
     * computer. This can be done by allowing ingress from this computer's IP
     * address. In some situations, such as connecting from a corporate network,
you
     * must instead specify a prefix list ID. You can also temporarily open the
port
     * to
     * any IP address while running this example. If you do, be sure to remove
     * public
     * access when you're done.
     *
     */
    public GroupInfo verifyInboundPort(String VPC, int port, String ipAddress) {
        boolean portIsOpen = false;
```

```
        GroupInfo groupInfo = new GroupInfo();
        try {
            Filter filter = Filter.builder()
                    .name("group-name")
                    .values("default")
                    .build();

            Filter filter1 = Filter.builder()
                    .name("vpc-id")
                    .values(VPC)
                    .build();

            DescribeSecurityGroupsRequest securityGroupsRequest =
DescribeSecurityGroupsRequest.builder()
                    .filters(filter, filter1)
                    .build();

            DescribeSecurityGroupsResponse securityGroupsResponse =
getEc2Client()
                    .describeSecurityGroups(securityGroupsRequest);
            String securityGroup =
securityGroupsResponse.securityGroups().get(0).groupName();
            groupInfo.setGroupName(securityGroup);

            for (SecurityGroup secGroup :
securityGroupsResponse.securityGroups()) {
                System.out.println("Found security group: " +
secGroup.groupId());

                for (IpPermission ipPermission : secGroup.ipPermissions()) {
                    if (ipPermission.fromPort() == port) {
                        System.out.println("Found inbound rule: " +
ipPermission);
                        for (IpRange ipRange : ipPermission.ipRanges()) {
                            String cidrIp = ipRange.cidrIp();
                            if (cidrIp.startsWith(ipAddress) ||
cidrIp.equals("0.0.0.0/0")) {
                                System.out.println(cidrIp + " is applicable");
                                portIsOpen = true;
                            }
                        }

                        if (!ipPermission.prefixListIds().isEmpty()) {
                            System.out.println("Prefix lList is applicable");
```

```
                            portIsOpen = true;
                        }

                        if (!portIsOpen) {
                            System.out
                                    .println("The inbound rule does not appear to
be open to either this computer's IP,"
                                            + " all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                        } else {
                            break;
                        }
                    }
                }
            }

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }

        groupInfo.setPortOpen(portIsOpen);
        return groupInfo;
    }

    /*
     * Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
     * Scaling group.
     * The target group specifies how the load balancer forward requests to the
     * instances
     * in the group.
     */
    public void attachLoadBalancerTargetGroup(String asGroupName, String
targetGroupARN) {
        try {
            AttachLoadBalancerTargetGroupsRequest targetGroupsRequest =
AttachLoadBalancerTargetGroupsRequest.builder()
                    .autoScalingGroupName(asGroupName)
                    .targetGroupARNs(targetGroupARN)
                    .build();


getAutoScalingClient().attachLoadBalancerTargetGroups(targetGroupsRequest);
            System.out.println("Attached load balancer to " + asGroupName);
```

```
        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    // Creates an EC2 Auto Scaling group with the specified size.
    public String[] createGroup(int groupSize, String templateName, String
 autoScalingGroupName) {

        // Get availability zones.

 software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesRequest
 zonesRequest =
 software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesRequest
                .builder()
                .build();

        DescribeAvailabilityZonesResponse zonesResponse =
 getEc2Client().describeAvailabilityZones(zonesRequest);
        List<String> availabilityZoneNames =
 zonesResponse.availabilityZones().stream()

 .map(software.amazon.awssdk.services.ec2.model.AvailabilityZone::zoneName)
                .collect(Collectors.toList());

        String availabilityZones = String.join(",", availabilityZoneNames);
        LaunchTemplateSpecification specification =
 LaunchTemplateSpecification.builder()
                .launchTemplateName(templateName)
                .version("$Default")
                .build();

        String[] zones = availabilityZones.split(",");
        CreateAutoScalingGroupRequest groupRequest =
 CreateAutoScalingGroupRequest.builder()
                .launchTemplate(specification)
                .availabilityZones(zones)
                .maxSize(groupSize)
                .minSize(groupSize)
                .autoScalingGroupName(autoScalingGroupName)
                .build();

        try {
```

```
            getAutoScalingClient().createAutoScalingGroup(groupRequest);

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Created an EC2 Auto Scaling group named " +
 autoScalingGroupName);
        return zones;
    }

    public String getDefaultVPC() {
        // Define the filter.
        Filter defaultFilter = Filter.builder()
                .name("is-default")
                .values("true")
                .build();

        software.amazon.awssdk.services.ec2.model.DescribeVpcsRequest request =
 software.amazon.awssdk.services.ec2.model.DescribeVpcsRequest
                .builder()
                .filters(defaultFilter)
                .build();

        DescribeVpcsResponse response = getEc2Client().describeVpcs(request);
        return response.vpcs().get(0).vpcId();
    }

    // Gets the default subnets in a VPC for a specified list of Availability
 Zones.
    public List<Subnet> getSubnets(String vpcId, String[] availabilityZones) {
        List<Subnet> subnets = null;
        Filter vpcFilter = Filter.builder()
                .name("vpc-id")
                .values(vpcId)
                .build();

        Filter azFilter = Filter.builder()
                .name("availability-zone")
                .values(availabilityZones)
                .build();

        Filter defaultForAZ = Filter.builder()
                .name("default-for-az")
```

```
                .values("true")
                .build();

        DescribeSubnetsRequest request = DescribeSubnetsRequest.builder()
                .filters(vpcFilter, azFilter, defaultForAZ)
                .build();

        DescribeSubnetsResponse response =
getEc2Client().describeSubnets(request);
        subnets = response.subnets();
        return subnets;
    }

    // Gets data about the instances in the EC2 Auto Scaling group.
    public String getBadInstance(String groupName) {
        DescribeAutoScalingGroupsRequest request =
DescribeAutoScalingGroupsRequest.builder()
                .autoScalingGroupNames(groupName)
                .build();

        DescribeAutoScalingGroupsResponse response =
getAutoScalingClient().describeAutoScalingGroups(request);
        AutoScalingGroup autoScalingGroup = response.autoScalingGroups().get(0);
        List<String> instanceIds = autoScalingGroup.instances().stream()
                .map(instance -> instance.instanceId())
                .collect(Collectors.toList());

        String[] instanceIdArray = instanceIds.toArray(new String[0]);
        for (String instanceId : instanceIdArray) {
            System.out.println("Instance ID: " + instanceId);
            return instanceId;
        }
        return "";
    }

    // Gets data about the profile associated with an instance.
    public String getInstanceProfile(String instanceId) {
        Filter filter = Filter.builder()
                .name("instance-id")
                .values(instanceId)
                .build();

        DescribeIamInstanceProfileAssociationsRequest associationsRequest =
DescribeIamInstanceProfileAssociationsRequest
```

```
                .builder()
                .filters(filter)
                .build();

        DescribeIamInstanceProfileAssociationsResponse response = getEc2Client()
                .describeIamInstanceProfileAssociations(associationsRequest);
        return response.iamInstanceProfileAssociations().get(0).associationId();
    }

    public void deleteRolesPolicies(String policyName, String roleName, String
 InstanceProfile) {
        ListPoliciesRequest listPoliciesRequest =
 ListPoliciesRequest.builder().build();
        ListPoliciesResponse listPoliciesResponse =
 getIAMClient().listPolicies(listPoliciesRequest);
        for (Policy policy : listPoliciesResponse.policies()) {
            if (policy.policyName().equals(policyName)) {
                // List the entities (users, groups, roles) that are attached to
 the policy.

 software.amazon.awssdk.services.iam.model.ListEntitiesForPolicyRequest
 listEntitiesRequest =
 software.amazon.awssdk.services.iam.model.ListEntitiesForPolicyRequest
                        .builder()
                        .policyArn(policy.arn())
                        .build();
                ListEntitiesForPolicyResponse listEntitiesResponse = iamClient
                        .listEntitiesForPolicy(listEntitiesRequest);
                if (!listEntitiesResponse.policyGroups().isEmpty() || !
listEntitiesResponse.policyUsers().isEmpty()
                        || !listEntitiesResponse.policyRoles().isEmpty()) {
                    // Detach the policy from any entities it is attached to.
                    DetachRolePolicyRequest detachPolicyRequest =
 DetachRolePolicyRequest.builder()
                            .policyArn(policy.arn())
                            .roleName(roleName) // Specify the name of the IAM
 role
                            .build();

                    getIAMClient().detachRolePolicy(detachPolicyRequest);
                    System.out.println("Policy detached from entities.");
                }

                // Now, you can delete the policy.
```

```
                DeletePolicyRequest deletePolicyRequest =
DeletePolicyRequest.builder()
                        .policyArn(policy.arn())
                        .build();

                getIAMClient().deletePolicy(deletePolicyRequest);
                System.out.println("Policy deleted successfully.");
                break;
            }
        }

        // List the roles associated with the instance profile
        ListInstanceProfilesForRoleRequest listRolesRequest =
ListInstanceProfilesForRoleRequest.builder()
                .roleName(roleName)
                .build();

        // Detach the roles from the instance profile
        ListInstanceProfilesForRoleResponse listRolesResponse =
iamClient.listInstanceProfilesForRole(listRolesRequest);
        for (software.amazon.awssdk.services.iam.model.InstanceProfile profile :
listRolesResponse.instanceProfiles()) {
            RemoveRoleFromInstanceProfileRequest removeRoleRequest =
RemoveRoleFromInstanceProfileRequest.builder()
                    .instanceProfileName(InstanceProfile)
                    .roleName(roleName) // Remove the extra dot here
                    .build();

            getIAMClient().removeRoleFromInstanceProfile(removeRoleRequest);
            System.out.println("Role " + roleName + " removed from instance
profile " + InstanceProfile);
        }

        // Delete the instance profile after removing all roles
        DeleteInstanceProfileRequest deleteInstanceProfileRequest =
DeleteInstanceProfileRequest.builder()
                .instanceProfileName(InstanceProfile)
                .build();

        getIAMClient().deleteInstanceProfile(r ->
r.instanceProfileName(InstanceProfile));
        System.out.println(InstanceProfile + " Deleted");
        System.out.println("All roles and policies are deleted.");
    }
```

```
}
```

Create a class that wraps Elastic Load Balancing actions.

```
public class LoadBalancer {
    public ElasticLoadBalancingV2Client elasticLoadBalancingV2Client;

    public ElasticLoadBalancingV2Client getLoadBalancerClient() {
        if (elasticLoadBalancingV2Client == null) {
            elasticLoadBalancingV2Client = ElasticLoadBalancingV2Client.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }

        return elasticLoadBalancingV2Client;
    }

    // Checks the health of the instances in the target group.
    public List<TargetHealthDescription> checkTargetHealth(String
 targetGroupName) {
        DescribeTargetGroupsRequest targetGroupsRequest =
 DescribeTargetGroupsRequest.builder()
                .names(targetGroupName)
                .build();

        DescribeTargetGroupsResponse tgResponse =
 getLoadBalancerClient().describeTargetGroups(targetGroupsRequest);

        DescribeTargetHealthRequest healthRequest =
 DescribeTargetHealthRequest.builder()

 .targetGroupArn(tgResponse.targetGroups().get(0).targetGroupArn())
                .build();

        DescribeTargetHealthResponse healthResponse =
 getLoadBalancerClient().describeTargetHealth(healthRequest);
        return healthResponse.targetHealthDescriptions();
    }

    // Gets the HTTP endpoint of the load balancer.
    public String getEndpoint(String lbName) {
        DescribeLoadBalancersResponse res = getLoadBalancerClient()
```

```
                .describeLoadBalancers(describe -> describe.names(lbName));
        return res.loadBalancers().get(0).dnsName();
    }

    // Deletes a load balancer.
    public void deleteLoadBalancer(String lbName) {
        try {
            // Use a waiter to delete the Load Balancer.
            DescribeLoadBalancersResponse res = getLoadBalancerClient()
                    .describeLoadBalancers(describe -> describe.names(lbName));
            ElasticLoadBalancingV2Waiter loadBalancerWaiter =
getLoadBalancerClient().waiter();
            DescribeLoadBalancersRequest request =
DescribeLoadBalancersRequest.builder()

.loadBalancerArns(res.loadBalancers().get(0).loadBalancerArn())
                    .build();

            getLoadBalancerClient().deleteLoadBalancer(
                    builder ->
builder.loadBalancerArn(res.loadBalancers().get(0).loadBalancerArn()));
            WaiterResponse<DescribeLoadBalancersResponse> waiterResponse =
loadBalancerWaiter
                    .waitUntilLoadBalancersDeleted(request);
            waiterResponse.matched().response().ifPresent(System.out::println);

        } catch (ElasticLoadBalancingV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
        System.out.println(lbName + " was deleted.");
    }

    // Deletes the target group.
    public void deleteTargetGroup(String targetGroupName) {
        try {
            DescribeTargetGroupsResponse res = getLoadBalancerClient()
                    .describeTargetGroups(describe ->
describe.names(targetGroupName));
            getLoadBalancerClient()
                    .deleteTargetGroup(builder ->
builder.targetGroupArn(res.targetGroups().get(0).targetGroupArn()));
        } catch (ElasticLoadBalancingV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
```

```
        System.out.println(targetGroupName + " was deleted.");
    }

    // Verify this computer can successfully send a GET request to the load
balancer
    // endpoint.
    public boolean verifyLoadBalancerEndpoint(String elbDnsName) throws
IOException, InterruptedException {
        boolean success = false;
        int retries = 3;
        CloseableHttpClient httpClient = HttpClients.createDefault();

        // Create an HTTP GET request to the ELB.
        HttpGet httpGet = new HttpGet("http://" + elbDnsName);
        try {
            while ((!success) && (retries > 0)) {
                // Execute the request and get the response.
                HttpResponse response = httpClient.execute(httpGet);
                int statusCode = response.getStatusLine().getStatusCode();
                System.out.println("HTTP Status Code: " + statusCode);
                if (statusCode == 200) {
                    success = true;
                } else {
                    retries--;
                    System.out.println("Got connection error from load balancer
endpoint, retrying...");
                    TimeUnit.SECONDS.sleep(15);
                }
            }

        } catch (org.apache.http.conn.HttpHostConnectException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("Status.." + success);
        return success;
    }

    /*
     * Creates an Elastic Load Balancing target group. The target group specifies
     * how
     * the load balancer forward requests to instances in the group and how
instance
     * health is checked.
```

```
    */
   public String createTargetGroup(String protocol, int port, String vpcId,
String targetGroupName) {
       CreateTargetGroupRequest targetGroupRequest =
CreateTargetGroupRequest.builder()
               .healthCheckPath("/healthcheck")
               .healthCheckTimeoutSeconds(5)
               .port(port)
               .vpcId(vpcId)
               .name(targetGroupName)
               .protocol(protocol)
               .build();

       CreateTargetGroupResponse targetGroupResponse =
getLoadBalancerClient().createTargetGroup(targetGroupRequest);
       String targetGroupArn =
targetGroupResponse.targetGroups().get(0).targetGroupArn();
       String targetGroup =
targetGroupResponse.targetGroups().get(0).targetGroupName();
       System.out.println("The " + targetGroup + " was created with ARN" +
targetGroupArn);
       return targetGroupArn;
   }

   /*
    * Creates an Elastic Load Balancing load balancer that uses the specified
    * subnets
    * and forwards requests to the specified target group.
    */
   public String createLoadBalancer(List<Subnet> subnetIds, String
targetGroupARN, String lbName, int port,
           String protocol) {
       try {
           List<String> subnetIdStrings = subnetIds.stream()
                   .map(Subnet::subnetId)
                   .collect(Collectors.toList());

           CreateLoadBalancerRequest balancerRequest =
CreateLoadBalancerRequest.builder()
                   .subnets(subnetIdStrings)
                   .name(lbName)
                   .scheme("internet-facing")
                   .build();
```

```
            // Create and wait for the load balancer to become available.
            CreateLoadBalancerResponse lsResponse =
getLoadBalancerClient().createLoadBalancer(balancerRequest);
            String lbARN = lsResponse.loadBalancers().get(0).loadBalancerArn();

            ElasticLoadBalancingV2Waiter loadBalancerWaiter =
getLoadBalancerClient().waiter();
            DescribeLoadBalancersRequest request =
DescribeLoadBalancersRequest.builder()
                    .loadBalancerArns(lbARN)
                    .build();

            System.out.println("Waiting for Load Balancer " + lbName + " to
become available.");
            WaiterResponse<DescribeLoadBalancersResponse> waiterResponse =
loadBalancerWaiter
                    .waitUntilLoadBalancerAvailable(request);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("Load Balancer " + lbName + " is available.");

            // Get the DNS name (endpoint) of the load balancer.
            String lbDNSName = lsResponse.loadBalancers().get(0).dnsName();
            System.out.println("*** Load Balancer DNS Name: " + lbDNSName);

            // Create a listener for the load balance.
            Action action = Action.builder()
                    .targetGroupArn(targetGroupARN)
                    .type("forward")
                    .build();

            CreateListenerRequest listenerRequest =
CreateListenerRequest.builder()

.loadBalancerArn(lsResponse.loadBalancers().get(0).loadBalancerArn())
                    .defaultActions(action)
                    .port(port)
                    .protocol(protocol)
                    .defaultActions(action)
                    .build();

            getLoadBalancerClient().createListener(listenerRequest);
            System.out.println("Created listener to forward traffic from load
balancer " + lbName + " to target group "
                    + targetGroupARN);
```

```
            // Return the load balancer DNS name.
            return lbDNSName;

        } catch (ElasticLoadBalancingV2Exception e) {
            e.printStackTrace();
        }
        return "";
    }
}
```

Create a class that uses DynamoDB to simulate a recommendation service.

```
public class Database {

    private static DynamoDbClient dynamoDbClient;

    public static DynamoDbClient getDynamoDbClient() {
        if (dynamoDbClient == null) {
            dynamoDbClient = DynamoDbClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return dynamoDbClient;
    }

    // Checks to see if the Amazon DynamoDB table exists.
    private boolean doesTableExist(String tableName) {
        try {
            // Describe the table and catch any exceptions.
            DescribeTableRequest describeTableRequest =
 DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            getDynamoDbClient().describeTable(describeTableRequest);
            System.out.println("Table '" + tableName + "' exists.");
            return true;

        } catch (ResourceNotFoundException e) {
            System.out.println("Table '" + tableName + "' does not exist.");
        } catch (DynamoDbException e) {
```

```
            System.err.println("Error checking table existence: " +
e.getMessage());
        }
        return false;
    }

    /*
     * Creates a DynamoDB table to use a recommendation service. The table has a
     * hash key named 'MediaType' that defines the type of media recommended,
such
     * as
     * Book or Movie, and a range key named 'ItemId' that, combined with the
     * MediaType,
     * forms a unique identifier for the recommended item.
     */
    public void createTable(String tableName, String fileName) throws IOException
{
        // First check to see if the table exists.
        boolean doesExist = doesTableExist(tableName);
        if (!doesExist) {
            DynamoDbWaiter dbWaiter = getDynamoDbClient().waiter();
            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                    .tableName(tableName)
                    .attributeDefinitions(
                            AttributeDefinition.builder()
                                    .attributeName("MediaType")
                                    .attributeType(ScalarAttributeType.S)
                                    .build(),
                            AttributeDefinition.builder()
                                    .attributeName("ItemId")
                                    .attributeType(ScalarAttributeType.N)
                                    .build())
                    .keySchema(
                            KeySchemaElement.builder()
                                    .attributeName("MediaType")
                                    .keyType(KeyType.HASH)
                                    .build(),
                            KeySchemaElement.builder()
                                    .attributeName("ItemId")
                                    .keyType(KeyType.RANGE)
                                    .build())
                    .provisionedThroughput(
                            ProvisionedThroughput.builder()
                                    .readCapacityUnits(5L)
```

```
                                    .writeCapacityUnits(5L)
                                    .build())
                    .build();

            getDynamoDbClient().createTable(createTableRequest);
            System.out.println("Creating table " + tableName + "...");

            // Wait until the Amazon DynamoDB table is created.
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("Table " + tableName + " created.");

            // Add records to the table.
            populateTable(fileName, tableName);
        }
    }

    public void deleteTable(String tableName) {
        getDynamoDbClient().deleteTable(table -> table.tableName(tableName));
        System.out.println("Table " + tableName + " deleted.");
    }

    // Populates the table with data located in a JSON file using the DynamoDB
    // enhanced client.
    public void populateTable(String fileName, String tableName) throws
IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(getDynamoDbClient())
                .build();
        ObjectMapper objectMapper = new ObjectMapper();
        File jsonFile = new File(fileName);
        JsonNode rootNode = objectMapper.readTree(jsonFile);

        DynamoDbTable<Recommendation> mappedTable =
enhancedClient.table(tableName,
                TableSchema.fromBean(Recommendation.class));
        for (JsonNode currentNode : rootNode) {
            String mediaType = currentNode.path("MediaType").path("S").asText();
            int itemId = currentNode.path("ItemId").path("N").asInt();
```

```
            String title = currentNode.path("Title").path("S").asText();
            String creator = currentNode.path("Creator").path("S").asText();

            // Create a Recommendation object and set its properties.
            Recommendation rec = new Recommendation();
            rec.setMediaType(mediaType);
            rec.setItemId(itemId);
            rec.setTitle(title);
            rec.setCreator(creator);

            // Put the item into the DynamoDB table.
            mappedTable.putItem(rec); // Add the Recommendation to the list.
        }
        System.out.println("Added all records to the " + tableName);
    }
}
```

Create a class that wraps Systems Manager actions.

```
public class ParameterHelper {

    String tableName = "doc-example-resilient-architecture-table";
    String dyntable = "doc-example-recommendation-service";
    String failureResponse = "doc-example-resilient-architecture-failure-
response";
    String healthCheck = "doc-example-resilient-architecture-health-check";

    public void reset() {
        put(dyntable, tableName);
        put(failureResponse, "none");
        put(healthCheck, "shallow");
    }

    public void put(String name, String value) {
        SsmClient ssmClient = SsmClient.builder()
                .region(Region.US_EAST_1)
                .build();

        PutParameterRequest parameterRequest = PutParameterRequest.builder()
                .name(name)
                .value(value)
                .overwrite(true)
```

```
                .type("String")
                .build();

        ssmClient.putParameter(parameterRequest);
        System.out.printf("Setting demo parameter %s to '%s'.", name, value);
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances

  - ReplaceIamInstanceProfileAssociation

- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
```

```
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Create steps to deploy all of the resources.

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
```

```
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
```

```
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
```

```
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item']
[] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
```

```
    new ScenarioOutput(
      "createdKeyPair",
      MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
    ),
    new ScenarioOutput(
      "creatingInstancePolicy",
      MESSAGES.creatingInstancePolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      ),
    ),
    new ScenarioAction("createInstancePolicy", async (state) => {
      const client = new IAMClient({});
      const {
        Policy: { Arn },
      } = await client.send(
        new CreatePolicyCommand({
          PolicyName: NAMES.instancePolicyName,
          PolicyDocument: readFileSync(
            join(RESOURCES_PATH, "instance_policy.json"),
          ),
        }),
      );
      state.instancePolicyArn = Arn;
    }),
    new ScenarioOutput("createdInstancePolicy", (state) =>
      MESSAGES.createdInstancePolicy
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
    ),
    new ScenarioOutput(
      "creatingInstanceRole",
      MESSAGES.creatingInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      ),
    ),
    new ScenarioAction("createInstanceRole", () => {
      const client = new IAMClient({});
      return client.send(
        new CreateRoleCommand({
          RoleName: NAMES.instanceRoleName,
          AssumeRolePolicyDocument: readFileSync(
            join(ROOT, "assume-role-policy.json"),
```

```
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
```

```
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
```

```
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
```

```
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
```

```
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
```

```
    }),
    new ScenarioOutput(
      "createdLoadBalancerTargetGroup",
      MESSAGES.createdLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      ),
    ),
    new ScenarioOutput(
      "creatingLoadBalancer",
      MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
    ),
    new ScenarioAction("createLoadBalancer", async (state) => {
      const client = new ElasticLoadBalancingV2Client({});
      const { LoadBalancers } = await client.send(
        new CreateLoadBalancerCommand({
          Name: NAMES.loadBalancerName,
          Subnets: state.subnets,
        }),
      );
      state.loadBalancerDns = LoadBalancers[0].DNSName;
      state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
      await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
      );
    }),
    new ScenarioOutput("createdLoadBalancer", (state) =>
      MESSAGES.createdLoadBalancer
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${DNS_NAME}", state.loadBalancerDns),
    ),
    new ScenarioOutput(
      "creatingListener",
      MESSAGES.creatingLoadBalancerListener
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
    ),
    new ScenarioAction("createListener", async (state) => {
      const client = new ElasticLoadBalancingV2Client({});
      const { Listeners } = await client.send(
        new CreateListenerCommand({
          LoadBalancerArn: state.loadBalancerArn,
          Protocol: state.targetGroupProtocol,
```

```
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup}} state
     */
    async (state) => {
      const client = new EC2Client({});
```

```
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
```

```
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
```

```
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

Create steps to run the demo.

```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
```

```
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
```

```
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
```

```
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
```

```
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
```

```
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-
scaling').Instance }} state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
```

```
    );
    state.targetInstance = AutoScalingGroups[0].Instances[0];
    const ec2Client = new EC2Client({});
    const { IamInstanceProfileAssociations } = await ec2Client.send(
      new DescribeIamInstanceProfileAssociationsCommand({
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      }),
    );
    state.instanceProfileAssociationId =
      IamInstanceProfileAssociations[0].AssociationId;
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        }),
      ),
    );

    await ec2Client.send(
      new RebootInstancesCommand({
        InstanceIds: [state.targetInstance.InstanceId],
      }),
    );

    const ssmClient = new SSMClient({});
    await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
      const { InstanceInformationList } = await ssmClient.send(
        new DescribeInstanceInformationCommand({}),
      );

      const instance = InstanceInformationList.find(
        (info) => info.InstanceId === state.targetInstance.InstanceId,
      );

      if (!instance) {
        throw new Error("Instance not found.");
      }
    });

    await ssmClient.send(
      new SendCommandCommand({
```

```
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation}} state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
```

```
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
```

```
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
```

```
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
```

```
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

Create steps to destroy all of the resources.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```
import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
```

```
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
```

```
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
```

```
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
```

```
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
```

```
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
```

```
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
```

```
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
```

```
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
```

```
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
```

```
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
 state,
    ) => {
```

```
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    } else {
      return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}
```

```
/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}
```

```
async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets

- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
class Runner:
    """
    Manages the deployment, demonstration, and destruction of resources for the
 resilient service.
    """

    def __init__(
        self,
        resource_path: str,
        recommendation: RecommendationService,
        autoscaler: AutoScalingWrapper,
        loadbalancer: ElasticLoadBalancerWrapper,
        param_helper: ParameterHelper,
    ):
        """
        Initializes the Runner class with the necessary parameters.

        :param resource_path: The path to resource files used by this example,
 such as IAM policies and instance scripts.
```

```
        :param recommendation: An instance of the RecommendationService class.
        :param autoscaler: An instance of the AutoScaler class.
        :param loadbalancer: An instance of the LoadBalancer class.
        :param param_helper: An instance of the ParameterHelper class.
        """
        self.resource_path = resource_path
        self.recommendation = recommendation
        self.autoscaler = autoscaler
        self.loadbalancer = loadbalancer
        self.param_helper = param_helper
        self.protocol = "HTTP"
        self.port = 80
        self.ssh_port = 22

        prefix = "doc-example-resilience"
        self.target_group_name = f"{prefix}-tg"
        self.load_balancer_name = f"{prefix}-lb"

    def deploy(self) -> None:
        """
        Deploys the resources required for the resilient service, including the
DynamoDB table,
        EC2 instances, Auto Scaling group, and load balancer.
        """
        recommendations_path = f"{self.resource_path}/recommendations.json"
        startup_script = f"{self.resource_path}/server_startup_script.sh"
        instance_policy = f"{self.resource_path}/instance_policy.json"

        logging.info("Starting deployment of resources for the resilient
service.")

        logging.info(
            "Creating and populating DynamoDB table '%s'.",
            self.recommendation.table_name,
        )
        self.recommendation.create()
        self.recommendation.populate(recommendations_path)

        logging.info(
            "Creating an EC2 launch template with the startup script '%s'.",
            startup_script,
        )
        self.autoscaler.create_template(startup_script, instance_policy)
```

```
        logging.info(
            "Creating an EC2 Auto Scaling group across multiple Availability
 Zones."
        )
        zones = self.autoscaler.create_autoscaling_group(3)

        logging.info("Creating variables that control the flow of the demo.")
        self.param_helper.reset()

        logging.info("Creating Elastic Load Balancing target group and load
 balancer.")

        vpc = self.autoscaler.get_default_vpc()
        subnets = self.autoscaler.get_subnets(vpc["VpcId"], zones)
        target_group = self.loadbalancer.create_target_group(
            self.target_group_name, self.protocol, self.port, vpc["VpcId"]
        )
        self.loadbalancer.create_load_balancer(
            self.load_balancer_name, [subnet["SubnetId"] for subnet in subnets]
        )
        self.loadbalancer.create_listener(self.load_balancer_name, target_group)

        self.autoscaler.attach_load_balancer_target_group(target_group)

        logging.info("Verifying access to the load balancer endpoint.")
        endpoint = self.loadbalancer.get_endpoint(self.load_balancer_name)
        lb_success = self.loadbalancer.verify_load_balancer_endpoint(endpoint)
        current_ip_address = requests.get("http://
checkip.amazonaws.com").text.strip()

        if not lb_success:
            logging.warning(
                "Couldn't connect to the load balancer. Verifying that the port
 is open..."
            )
            sec_group, port_is_open = self.autoscaler.verify_inbound_port(
                vpc, self.port, current_ip_address
            )
            sec_group, ssh_port_is_open = self.autoscaler.verify_inbound_port(
                vpc, self.ssh_port, current_ip_address
            )
            if not port_is_open:
                logging.warning(
```

```
                    "The default security group for your VPC must allow access
from this computer."
                )
                if q.ask(
                    f"Do you want to add a rule to security group
{sec_group['GroupId']} to allow\n"
                    f"inbound traffic on port {self.port} from your computer's IP
address of {current_ip_address}? (y/n) ",
                    q.is_yesno,
                ):
                    self.autoscaler.open_inbound_port(
                        sec_group["GroupId"], self.port, current_ip_address
                    )
            if not ssh_port_is_open:
                if q.ask(
                    f"Do you want to add a rule to security group
{sec_group['GroupId']} to allow\n"
                    f"inbound SSH traffic on port {self.ssh_port} for debugging
from your computer's IP address of {current_ip_address}? (y/n) ",
                    q.is_yesno,
                ):
                    self.autoscaler.open_inbound_port(
                        sec_group["GroupId"], self.ssh_port, current_ip_address
                    )
            lb_success =
self.loadbalancer.verify_load_balancer_endpoint(endpoint)

        if lb_success:
            logging.info(
                "Load balancer is ready. Access it at: http://%s",
current_ip_address
            )
        else:
            logging.error(
                "Couldn't get a successful response from the load balancer
endpoint. Please verify your VPC and security group settings."
            )

    def demo_choices(self) -> None:
        """
        Presents choices for interacting with the deployed service, such as
sending requests to
        the load balancer or checking the health of the targets.
        """
```

```
        actions = [
            "Send a GET request to the load balancer endpoint.",
            "Check the health of load balancer targets.",
            "Go to the next part of the demo.",
        ]
        choice = 0
        while choice != 2:
            logging.info("Choose an action to interact with the service.")
            choice = q.choose("Which action would you like to take? ", actions)
            if choice == 0:
                logging.info("Sending a GET request to the load balancer
endpoint.")
                endpoint =
self.loadbalancer.get_endpoint(self.load_balancer_name)
                logging.info("GET http://%s", endpoint)
                response = requests.get(f"http://{endpoint}")
                logging.info("Response: %s", response.status_code)
                if response.headers.get("content-type") == "application/json":
                    pp(response.json())
            elif choice == 1:
                logging.info("Checking the health of load balancer targets.")
                health =
self.loadbalancer.check_target_health(self.target_group_name)
                for target in health:
                    state = target["TargetHealth"]["State"]
                    logging.info(
                        "Target %s on port %d is %s",
                        target["Target"]["Id"],
                        target["Target"]["Port"],
                        state,
                    )
                    if state != "healthy":
                        logging.warning(
                            "%s: %s",
                            target["TargetHealth"]["Reason"],
                            target["TargetHealth"]["Description"],
                        )
                logging.info(
                    "Note that it can take a minute or two for the health check
to update."
                )
            elif choice == 2:
                logging.info("Proceeding to the next part of the demo.")
```

```
    def demo(self) -> None:
        """
        Runs the demonstration, showing how the service responds to different
failure scenarios
        and how a resilient architecture can keep the service running.
        """
        ssm_only_policy = f"{self.resource_path}/ssm_only_policy.json"

        logging.info("Resetting parameters to starting values for the demo.")
        self.param_helper.reset()

        logging.info(
            "Starting demonstration of the service's resilience under various
failure conditions."
        )
        self.demo_choices()

        logging.info(
            "Simulating failure by changing the Systems Manager parameter to a
non-existent table."
        )
        self.param_helper.put(self.param_helper.table, "this-is-not-a-table")
        logging.info("Sending GET requests will now return failure codes.")
        self.demo_choices()

        logging.info("Switching to static response mode to mitigate failure.")
        self.param_helper.put(self.param_helper.failure_response, "static")
        logging.info("Sending GET requests will now return static responses.")
        self.demo_choices()

        logging.info("Restoring normal operation of the recommendation service.")
        self.param_helper.put(self.param_helper.table,
self.recommendation.table_name)

        logging.info(
            "Introducing a failure by assigning bad credentials to one of the
instances."
        )
        self.autoscaler.create_instance_profile(
            ssm_only_policy,
            self.autoscaler.bad_creds_policy_name,
            self.autoscaler.bad_creds_role_name,
            self.autoscaler.bad_creds_profile_name,
            ["AmazonSSMManagedInstanceCore"],
```

```
        )
        instances = self.autoscaler.get_instances()
        bad_instance_id = instances[0]
        instance_profile = self.autoscaler.get_instance_profile(bad_instance_id)
        logging.info(
            "Replacing instance profile with bad credentials for instance %s.",
            bad_instance_id,
        )
        self.autoscaler.replace_instance_profile(
            bad_instance_id,
            self.autoscaler.bad_creds_profile_name,
            instance_profile["AssociationId"],
        )
        logging.info(
            "Sending GET requests may return either a valid recommendation or a
static response."
        )
        self.demo_choices()

        logging.info("Implementing deep health checks to detect unhealthy
instances.")
        self.param_helper.put(self.param_helper.health_check, "deep")
        logging.info("Checking the health of the load balancer targets.")
        self.demo_choices()

        logging.info(
            "Terminating the unhealthy instance to let the auto scaler replace
it."
        )
        self.autoscaler.terminate_instance(bad_instance_id)
        logging.info("The service remains resilient during instance
replacement.")
        self.demo_choices()

        logging.info("Simulating a complete failure of the recommendation
service.")
        self.param_helper.put(self.param_helper.table, "this-is-not-a-table")
        logging.info(
            "All instances will report as unhealthy, but the service will still
return static responses."
        )
        self.demo_choices()
        self.param_helper.reset()
```

```
    def destroy(self, automation=False) -> None:
        """
        Destroys all resources created for the demo, including the load balancer,
 Auto Scaling group,
        EC2 instances, and DynamoDB table.
        """
        logging.info(
            "This concludes the demo. Preparing to clean up all AWS resources
 created during the demo."
        )
        if automation:
            cleanup = True
        else:
            cleanup = q.ask(
                "Do you want to clean up all demo resources? (y/n) ", q.is_yesno
            )

        if cleanup:
            logging.info("Deleting load balancer and related resources.")
            self.loadbalancer.delete_load_balancer(self.load_balancer_name)
            self.loadbalancer.delete_target_group(self.target_group_name)
            self.autoscaler.delete_autoscaling_group(self.autoscaler.group_name)
            self.autoscaler.delete_key_pair()
            self.autoscaler.delete_template()
            self.autoscaler.delete_instance_profile(
                self.autoscaler.bad_creds_profile_name,
                self.autoscaler.bad_creds_role_name,
            )
            logging.info("Deleting DynamoDB table and other resources.")
            self.recommendation.destroy()
        else:
            logging.warning(
                "Resources have not been deleted. Ensure you clean them up
 manually to avoid unexpected charges."
            )


def main() -> None:
    """
    Main function to parse arguments and run the appropriate actions for the
 demo.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
```

```
        "--action",
        required=True,
        choices=["all", "deploy", "demo", "destroy"],
        help="The action to take for the demo. When 'all' is specified, resources
are\n"
        "deployed, the demo is run, and resources are destroyed.",
    )
    parser.add_argument(
        "--resource_path",
        default="../../../workflows/resilient_service/resources",
        help="The path to resource files used by this example, such as IAM
policies and\n"
        "instance scripts.",
    )
    args = parser.parse_args()

    logging.info("Starting the Resilient Service demo.")

    prefix = "doc-example-resilience"

    # Service Clients
    ddb_client = boto3.client("dynamodb")
    elb_client = boto3.client("elbv2")
    autoscaling_client = boto3.client("autoscaling")
    ec2_client = boto3.client("ec2")
    ssm_client = boto3.client("ssm")
    iam_client = boto3.client("iam")

    # Wrapper instantiations
    recommendation = RecommendationService(
        "doc-example-recommendation-service", ddb_client
    )
    autoscaling_wrapper = AutoScalingWrapper(
        prefix,
        "t3.micro",
        "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
        autoscaling_client,
        ec2_client,
        ssm_client,
        iam_client,
    )
    elb_wrapper = ElasticLoadBalancerWrapper(elb_client)
    param_helper = ParameterHelper(recommendation.table_name, ssm_client)
```

```
    # Demo invocation
    runner = Runner(
        args.resource_path,
        recommendation,
        autoscaling_wrapper,
        elb_wrapper,
        param_helper,
    )
    actions = [args.action] if args.action != "all" else ["deploy", "demo",
 "destroy"]
    for action in actions:
        if action == "deploy":
            runner.deploy()
        elif action == "demo":
            runner.demo()
        elif action == "destroy":
            runner.destroy()

    logging.info("Demo completed successfully.")


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    main()
```

Create a class that wraps Auto Scaling and Amazon EC2 actions.

```
class AutoScalingWrapper:
    """
    Encapsulates Amazon EC2 Auto Scaling and EC2 management actions.
    """

    def __init__(
        self,
        resource_prefix: str,
        inst_type: str,
        ami_param: str,
        autoscaling_client: boto3.client,
        ec2_client: boto3.client,
        ssm_client: boto3.client,
        iam_client: boto3.client,
    ):
```

```
        """
        Initializes the AutoScaler class with the necessary parameters.

        :param resource_prefix: The prefix for naming AWS resources that are
created by this class.
        :param inst_type: The type of EC2 instance to create, such as t3.micro.
        :param ami_param: The Systems Manager parameter used to look up the AMI
that is created.
        :param autoscaling_client: A Boto3 EC2 Auto Scaling client.
        :param ec2_client: A Boto3 EC2 client.
        :param ssm_client: A Boto3 Systems Manager client.
        :param iam_client: A Boto3 IAM client.
        """
        self.inst_type = inst_type
        self.ami_param = ami_param
        self.autoscaling_client = autoscaling_client
        self.ec2_client = ec2_client
        self.ssm_client = ssm_client
        self.iam_client = iam_client
        sts_client = boto3.client("sts")
        self.account_id = sts_client.get_caller_identity()["Account"]

        self.key_pair_name = f"{resource_prefix}-key-pair"
        self.launch_template_name = f"{resource_prefix}-template-"
        self.group_name = f"{resource_prefix}-group"

        # Happy path
        self.instance_policy_name = f"{resource_prefix}-pol"
        self.instance_role_name = f"{resource_prefix}-role"
        self.instance_profile_name = f"{resource_prefix}-prof"

        # Failure mode
        self.bad_creds_policy_name = f"{resource_prefix}-bc-pol"
        self.bad_creds_role_name = f"{resource_prefix}-bc-role"
        self.bad_creds_profile_name = f"{resource_prefix}-bc-prof"


    def create_policy(self, policy_file: str, policy_name: str) -> str:
        """
        Creates a new IAM policy or retrieves the ARN of an existing policy.

        :param policy_file: The path to a JSON file that contains the policy
definition.
        :param policy_name: The name to give the created policy.
```

```
        :return: The ARN of the created or existing policy.
        """
        with open(policy_file) as file:
            policy_doc = file.read()

        try:
            response = self.iam_client.create_policy(
                PolicyName=policy_name, PolicyDocument=policy_doc
            )
            policy_arn = response["Policy"]["Arn"]
            log.info(f"Policy '{policy_name}' created successfully. ARN:
 {policy_arn}")
            return policy_arn

        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                # If the policy already exists, get its ARN
                response = self.iam_client.get_policy(
                    PolicyArn=f"arn:aws:iam::{self.account_id}:policy/
{policy_name}"
                )
                policy_arn = response["Policy"]["Arn"]
                log.info(f"Policy '{policy_name}' already exists. ARN:
 {policy_arn}")
                return policy_arn
            log.error(f"Full error:\n\t{err}")

    def create_role(self, role_name: str, assume_role_doc: dict) -> str:
        """
        Creates a new IAM role or retrieves the ARN of an existing role.

        :param role_name: The name to give the created role.
        :param assume_role_doc: The assume role policy document that specifies
 which
                                entities can assume the role.
        :return: The ARN of the created or existing role.
        """
        try:
            response = self.iam_client.create_role(
                RoleName=role_name,
 AssumeRolePolicyDocument=json.dumps(assume_role_doc)
            )
            role_arn = response["Role"]["Arn"]
            log.info(f"Role '{role_name}' created successfully. ARN: {role_arn}")
```

```
            return role_arn

        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                # If the role already exists, get its ARN
                response = self.iam_client.get_role(RoleName=role_name)
                role_arn = response["Role"]["Arn"]
                log.info(f"Role '{role_name}' already exists. ARN: {role_arn}")
                return role_arn
            log.error(f"Full error:\n\t{err}")

    def attach_policy(
        self,
        role_name: str,
        policy_arn: str,
        aws_managed_policies: Tuple[str, ...] = (),
    ) -> None:
        """
        Attaches an IAM policy to a role and optionally attaches additional AWS-
managed policies.

        :param role_name: The name of the role to attach the policy to.
        :param policy_arn: The ARN of the policy to attach.
        :param aws_managed_policies: A tuple of AWS-managed policy names to
 attach to the role.
        """
        try:
            self.iam_client.attach_role_policy(RoleName=role_name,
 PolicyArn=policy_arn)
            for aws_policy in aws_managed_policies:
                self.iam_client.attach_role_policy(
                    RoleName=role_name,
                    PolicyArn=f"arn:aws:iam::aws:policy/{aws_policy}",
                )
            log.info(f"Attached policy {policy_arn} to role {role_name}.")
        except ClientError as err:
            log.error(f"Failed to attach policy {policy_arn} to role
 {role_name}.")
            log.error(f"Full error:\n\t{err}")

    def create_instance_profile(
        self,
        policy_file: str,
        policy_name: str,
```

```
        role_name: str,
        profile_name: str,
        aws_managed_policies: Tuple[str, ...] = (),
    ) -> str:
        """
        Creates a policy, role, and profile that is associated with instances
created by
        this class. An instance's associated profile defines a role that is
assumed by the
        instance. The role has attached policies that specify the AWS permissions
granted to
        clients that run on the instance.

        :param policy_file: The name of a JSON file that contains the policy
definition to
                            create and attach to the role.
        :param policy_name: The name to give the created policy.
        :param role_name: The name to give the created role.
        :param profile_name: The name to the created profile.
        :param aws_managed_policies: Additional AWS-managed policies that are
attached to
                                     the role, such as
AmazonSSMManagedInstanceCore to grant
                                     use of Systems Manager to send commands to
the instance.
        :return: The ARN of the profile that is created.
        """
        assume_role_doc = {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {"Service": "ec2.amazonaws.com"},
                    "Action": "sts:AssumeRole",
                }
            ],
        }
        policy_arn = self.create_policy(policy_file, policy_name)
        self.create_role(role_name, assume_role_doc)
        self.attach_policy(role_name, policy_arn, aws_managed_policies)

        try:
            profile_response = self.iam_client.create_instance_profile(
                InstanceProfileName=profile_name
```

```
            )
            waiter = self.iam_client.get_waiter("instance_profile_exists")
            waiter.wait(InstanceProfileName=profile_name)
            time.sleep(10)  # wait a little longer
            profile_arn = profile_response["InstanceProfile"]["Arn"]
            self.iam_client.add_role_to_instance_profile(
                InstanceProfileName=profile_name, RoleName=role_name
            )
            log.info("Created profile %s and added role %s.", profile_name,
role_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                prof_response = self.iam_client.get_instance_profile(
                    InstanceProfileName=profile_name
                )
                profile_arn = prof_response["InstanceProfile"]["Arn"]
                log.info(
                    "Instance profile %s already exists, nothing to do.",
profile_name
                )
            log.error(f"Full error:\n\t{err}")
        return profile_arn


    def get_instance_profile(self, instance_id: str) -> Dict[str, Any]:
        """
        Gets data about the profile associated with an instance.

        :param instance_id: The ID of the instance to look up.
        :return: The profile data.
        """
        try:
            response =
self.ec2_client.describe_iam_instance_profile_associations(
                Filters=[{"Name": "instance-id", "Values": [instance_id]}]
            )
            if not response["IamInstanceProfileAssociations"]:
                log.info(f"No instance profile found for instance
{instance_id}.")
            profile_data = response["IamInstanceProfileAssociations"][0]
            log.info(f"Retrieved instance profile for instance {instance_id}.")
            return profile_data
        except ClientError as err:
            log.error(
```

```
                f"Failed to retrieve instance profile for instance
{instance_id}."
            )
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidInstanceID.NotFound":
                log.error(f"The instance ID '{instance_id}' does not exist.")
            log.error(f"Full error:\n\t{err}")


    def replace_instance_profile(
        self,
        instance_id: str,
        new_instance_profile_name: str,
        profile_association_id: str,
    ) -> None:
        """
        Replaces the profile associated with a running instance. After the
profile is
        replaced, the instance is rebooted to ensure that it uses the new
profile. When
        the instance is ready, Systems Manager is used to restart the Python web
server.

        :param instance_id: The ID of the instance to restart.
        :param new_instance_profile_name: The name of the new profile to
associate with
                                          the specified instance.
        :param profile_association_id: The ID of the existing profile association
for the
                                       instance.
        """
        try:
            self.ec2_client.replace_iam_instance_profile_association(
                IamInstanceProfile={"Name": new_instance_profile_name},
                AssociationId=profile_association_id,
            )
            log.info(
                "Replaced instance profile for association %s with profile %s.",
                profile_association_id,
                new_instance_profile_name,
            )
            time.sleep(5)

            self.ec2_client.reboot_instances(InstanceIds=[instance_id])
```

```
            log.info("Rebooting instance %s.", instance_id)
            waiter = self.ec2_client.get_waiter("instance_running")
            log.info("Waiting for instance %s to be running.", instance_id)
            waiter.wait(InstanceIds=[instance_id])
            log.info("Instance %s is now running.", instance_id)

            self.ssm_client.send_command(
                InstanceIds=[instance_id],
                DocumentName="AWS-RunShellScript",
                Parameters={"commands": ["cd / && sudo python3 server.py 80"]},
            )
            log.info(f"Restarted the Python web server on instance
'{instance_id}'.")
        except ClientError as err:
            log.error("Failed to replace instance profile.")
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidAssociationID.NotFound":
                log.error(
                    f"Association ID '{profile_association_id}' does not exist."
                    "Please check the association ID and try again."
                )
            if error_code == "InvalidInstanceId":
                log.error(
                    f"The specified instance ID '{instance_id}' does not exist or
is not available for SSM. "
                    f"Please verify the instance ID and try again."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_instance_profile(self, profile_name: str, role_name: str) -> None:
        """
        Detaches a role from an instance profile, detaches policies from the
role,
        and deletes all the resources.

        :param profile_name: The name of the profile to delete.
        :param role_name: The name of the role to delete.
        """
        try:
            self.iam_client.remove_role_from_instance_profile(
                InstanceProfileName=profile_name, RoleName=role_name
            )
```

```
self.iam_client.delete_instance_profile(InstanceProfileName=profile_name)
            log.info("Deleted instance profile %s.", profile_name)
            attached_policies = self.iam_client.list_attached_role_policies(
                RoleName=role_name
            )
            for pol in attached_policies["AttachedPolicies"]:
                self.iam_client.detach_role_policy(
                    RoleName=role_name, PolicyArn=pol["PolicyArn"]
                )
                if not pol["PolicyArn"].startswith("arn:aws:iam::aws"):
                    self.iam_client.delete_policy(PolicyArn=pol["PolicyArn"])
                log.info("Detached and deleted policy %s.", pol["PolicyName"])
            self.iam_client.delete_role(RoleName=role_name)
            log.info("Deleted role %s.", role_name)
        except ClientError as err:
            log.error(
                f"Couldn't delete instance profile {profile_name} or detach "
                f"policies and delete role {role_name}: {err}"
            )
            if err.response["Error"]["Code"] == "NoSuchEntity":
                log.info(
                    "Instance profile %s doesn't exist, nothing to do.",
profile_name
                )


    def create_key_pair(self, key_pair_name: str) -> None:
        """
        Creates a new key pair.

        :param key_pair_name: The name of the key pair to create.
        """
        try:
            response = self.ec2_client.create_key_pair(KeyName=key_pair_name)
            with open(f"{key_pair_name}.pem", "w") as file:
                file.write(response["KeyMaterial"])
            chmod(f"{key_pair_name}.pem", 0o600)
            log.info("Created key pair %s.", key_pair_name)
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to create key pair {key_pair_name}.")
            if error_code == "InvalidKeyPair.Duplicate":
```

```
                log.error(f"A key pair with the name '{key_pair_name}' already
exists.")
            log.error(f"Full error:\n\t{err}")


    def delete_key_pair(self) -> None:
        """
        Deletes a key pair.
        """
        try:
            self.ec2_client.delete_key_pair(KeyName=self.key_pair_name)
            remove(f"{self.key_pair_name}.pem")
            log.info("Deleted key pair %s.", self.key_pair_name)
        except ClientError as err:
            log.error(f"Couldn't delete key pair '{self.key_pair_name}'.")
            log.error(f"Full error:\n\t{err}")
        except FileNotFoundError as err:
            log.info("Key pair %s doesn't exist, nothing to do.",
self.key_pair_name)
            log.error(f"Full error:\n\t{err}")


    def create_template(
        self, server_startup_script_file: str, instance_policy_file: str
    ) -> Dict[str, Any]:
        """
        Creates an Amazon EC2 launch template to use with Amazon EC2 Auto
Scaling. The
        launch template specifies a Bash script in its user data field that runs
after
        the instance is started. This script installs Python packages and starts
a
        Python web server on the instance.

        :param server_startup_script_file: The path to a Bash script file that is
run
                                           when an instance starts.
        :param instance_policy_file: The path to a file that defines a
permissions policy
                                     to create and attach to the instance
profile.
        :return: Information about the newly created template.
        """
        template = {}
```

```
        try:
            # Create key pair and instance profile
            self.create_key_pair(self.key_pair_name)
            self.create_instance_profile(
                instance_policy_file,
                self.instance_policy_name,
                self.instance_role_name,
                self.instance_profile_name,
            )

            # Read the startup script
            with open(server_startup_script_file) as file:
                start_server_script = file.read()

            # Get the latest AMI ID
            ami_latest = self.ssm_client.get_parameter(Name=self.ami_param)
            ami_id = ami_latest["Parameter"]["Value"]

            # Create the launch template
            lt_response = self.ec2_client.create_launch_template(
                LaunchTemplateName=self.launch_template_name,
                LaunchTemplateData={
                    "InstanceType": self.inst_type,
                    "ImageId": ami_id,
                    "IamInstanceProfile": {"Name": self.instance_profile_name},
                    "UserData": base64.b64encode(
                        start_server_script.encode(encoding="utf-8")
                    ).decode(encoding="utf-8"),
                    "KeyName": self.key_pair_name,
                },
            )
            template = lt_response["LaunchTemplate"]
            log.info(
                f"Created launch template {self.launch_template_name} for AMI
{ami_id} on {self.inst_type}."
            )
        except ClientError as err:
            log.error(f"Failed to create launch template
{self.launch_template_name}.")
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidLaunchTemplateName.AlreadyExistsException":
                log.info(
                    f"Launch template {self.launch_template_name} already exists,
nothing to do."
```

```
                )
            log.error(f"Full error:\n\t{err}")
        return template


    def delete_template(self):
        """
        Deletes a launch template.
        """
        try:
            self.ec2_client.delete_launch_template(
                LaunchTemplateName=self.launch_template_name
            )
            self.delete_instance_profile(
                self.instance_profile_name, self.instance_role_name
            )
            log.info("Launch template %s deleted.", self.launch_template_name)
        except ClientError as err:
            if (
                err.response["Error"]["Code"]
                == "InvalidLaunchTemplateName.NotFoundException"
            ):
                log.info(
                    "Launch template %s does not exist, nothing to do.",
                    self.launch_template_name,
                )
            log.error(f"Full error:\n\t{err}")


    def get_availability_zones(self) -> List[str]:
        """
        Gets a list of Availability Zones in the AWS Region of the Amazon EC2
client.

        :return: The list of Availability Zones for the client Region.
        """
        try:
            response = self.ec2_client.describe_availability_zones()
            zones = [zone["ZoneName"] for zone in response["AvailabilityZones"]]
            log.info(f"Retrieved {len(zones)} availability zones: {zones}.")
        except ClientError as err:
            log.error("Failed to retrieve availability zones.")
            log.error(f"Full error:\n\t{err}")
        else:
```

```
            return zones


    def create_autoscaling_group(self, group_size: int) -> List[str]:
        """
        Creates an EC2 Auto Scaling group with the specified size.

        :param group_size: The number of instances to set for the minimum and
maximum in
                           the group.
        :return: The list of Availability Zones specified for the group.
        """
        try:
            zones = self.get_availability_zones()
            self.autoscaling_client.create_auto_scaling_group(
                AutoScalingGroupName=self.group_name,
                AvailabilityZones=zones,
                LaunchTemplate={
                    "LaunchTemplateName": self.launch_template_name,
                    "Version": "$Default",
                },
                MinSize=group_size,
                MaxSize=group_size,
            )
            log.info(
                f"Created EC2 Auto Scaling group {self.group_name} with
availability zones {zones}."
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            if error_code == "AlreadyExists":
                log.info(
                    f"EC2 Auto Scaling group {self.group_name} already exists,
nothing to do."
                )
            else:
                log.error(f"Failed to create EC2 Auto Scaling group
{self.group_name}.")
                log.error(f"Full error:\n\t{err}")
        else:
            return zones


    def get_instances(self) -> List[str]:
```

```
        """
        Gets data about the instances in the EC2 Auto Scaling group.

        :return: A list of instance IDs in the Auto Scaling group.
        """
        try:
            as_response = self.autoscaling_client.describe_auto_scaling_groups(
                AutoScalingGroupNames=[self.group_name]
            )
            instance_ids = [
                i["InstanceId"]
                for i in as_response["AutoScalingGroups"][0]["Instances"]
            ]
            log.info(
                f"Retrieved {len(instance_ids)} instances for Auto Scaling group
{self.group_name}."
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to retrieve instances for Auto Scaling group
{self.group_name}."
            )
            if error_code == "ResourceNotFound":
                log.error(f"The Auto Scaling group '{self.group_name}' does not
exist.")
            log.error(f"Full error:\n\t{err}")
        else:
            return instance_ids


    def terminate_instance(self, instance_id: str, decrementsetting=False) ->
None:
        """
        Terminates an instance in an EC2 Auto Scaling group. After an instance is
        terminated, it can no longer be accessed.

        :param instance_id: The ID of the instance to terminate.
        :param decrementsetting: If True, do not replace terminated instances.
        """
        try:
            self.autoscaling_client.terminate_instance_in_auto_scaling_group(
                InstanceId=instance_id,
                ShouldDecrementDesiredCapacity=decrementsetting,
```

```
            )
            log.info("Terminated instance %s.", instance_id)

            # Adding a waiter to ensure the instance is terminated
            waiter = self.ec2_client.get_waiter("instance_terminated")
            log.info("Waiting for instance %s to be terminated...", instance_id)
            waiter.wait(InstanceIds=[instance_id])
            log.info(
                f"Instance '{instance_id}' has been terminated and will be
replaced."
            )

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to terminate instance '{instance_id}'.")
            if error_code == "ScalingActivityInProgressFault":
                log.error(
                    "Scaling activity is currently in progress. "
                    "Wait for the scaling activity to complete before attempting
to terminate the instance again."
                )
            elif error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
the resource."
                )
            log.error(f"Full error:\n\t{err}")

    def attach_load_balancer_target_group(
        self, lb_target_group: Dict[str, Any]
    ) -> None:
        """
        Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
        The target group specifies how the load balancer forwards requests to the
instances
        in the group.

        :param lb_target_group: Data about the ELB target group to attach.
        """
        try:
            self.autoscaling_client.attach_load_balancer_target_groups(
                AutoScalingGroupName=self.group_name,
```

```
                TargetGroupARNs=[lb_target_group["TargetGroupArn"]],
            )
            log.info(
                "Attached load balancer target group %s to auto scaling group
%s.",
                lb_target_group["TargetGroupName"],
                self.group_name,
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to attach load balancer target group
'{lb_target_group['TargetGroupName']}'."
            )
            if error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
the resource."
                )
            elif error_code == "ServiceLinkedRoleFailure":
                log.error(
                    "The operation failed because the service-linked role is not
ready or does not exist. "
                    "Check that the service-linked role exists and is correctly
configured."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_autoscaling_group(self, group_name: str) -> None:
        """
        Terminates all instances in the group, then deletes the EC2 Auto Scaling
group.

        :param group_name: The name of the group to delete.
        """
        try:
            response = self.autoscaling_client.describe_auto_scaling_groups(
                AutoScalingGroupNames=[group_name]
            )
            groups = response.get("AutoScalingGroups", [])
            if len(groups) > 0:
                self.autoscaling_client.update_auto_scaling_group(
```

```
                    AutoScalingGroupName=group_name, MinSize=0
                )
                instance_ids = [inst["InstanceId"] for inst in groups[0]
["Instances"]]
                for inst_id in instance_ids:
                    self.terminate_instance(inst_id)

                # Wait for all instances to be terminated
                if instance_ids:
                    waiter = self.ec2_client.get_waiter("instance_terminated")
                    log.info("Waiting for all instances to be terminated...")
                    waiter.wait(InstanceIds=instance_ids)
                    log.info("All instances have been terminated.")
            else:
                log.info(f"No groups found named '{group_name}'! Nothing to do.")
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to delete Auto Scaling group '{group_name}'.")
            if error_code == "ScalingActivityInProgressFault":
                log.error(
                    "Scaling activity is currently in progress. "
                    "Wait for the scaling activity to complete before attempting
 to delete the group again."
                )
            elif error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
 the group."
                )
            log.error(f"Full error:\n\t{err}")


    def get_default_vpc(self) -> Dict[str, Any]:
        """
        Gets the default VPC for the account.

        :return: Data about the default VPC.
        """
        try:
            response = self.ec2_client.describe_vpcs(
                Filters=[{"Name": "is-default", "Values": ["true"]}]
            )
        except ClientError as err:
```

```
            error_code = err.response["Error"]["Code"]
            log.error("Failed to retrieve the default VPC.")
            if error_code == "UnauthorizedOperation":
                log.error(
                    "You do not have the necessary permissions to describe VPCs.
 "
                    "Ensure that your AWS IAM user or role has the correct
 permissions."
                )
            elif error_code == "InvalidParameterValue":
                log.error(
                    "One or more parameters are invalid. Check the request
 parameters."
                )

            log.error(f"Full error:\n\t{err}")
        else:
            if "Vpcs" in response and response["Vpcs"]:
                log.info(f"Retrieved default VPC: {response['Vpcs'][0]
['VpcId']}")
                return response["Vpcs"][0]
            else:
                pass


    def verify_inbound_port(
        self, vpc: Dict[str, Any], port: int, ip_address: str
    ) -> Tuple[Dict[str, Any], bool]:
        """
        Verify the default security group of the specified VPC allows ingress
 from this
        computer. This can be done by allowing ingress from this computer's IP
        address. In some situations, such as connecting from a corporate network,
 you
        must instead specify a prefix list ID. You can also temporarily open the
 port to
        any IP address while running this example. If you do, be sure to remove
 public
        access when you're done.

        :param vpc: The VPC used by this example.
        :param port: The port to verify.
        :param ip_address: This computer's IP address.
```

```
        :return: The default security group of the specified VPC, and a value
that indicates
                 whether the specified port is open.
        """
        try:
            response = self.ec2_client.describe_security_groups(
                Filters=[
                    {"Name": "group-name", "Values": ["default"]},
                    {"Name": "vpc-id", "Values": [vpc["VpcId"]]},
                ]
            )
            sec_group = response["SecurityGroups"][0]
            port_is_open = False
            log.info(f"Found default security group {sec_group['GroupId']}.")

            for ip_perm in sec_group["IpPermissions"]:
                if ip_perm.get("FromPort", 0) == port:
                    log.info(f"Found inbound rule: {ip_perm}")
                    for ip_range in ip_perm["IpRanges"]:
                        cidr = ip_range.get("CidrIp", "")
                        if cidr.startswith(ip_address) or cidr == "0.0.0.0/0":
                            port_is_open = True
                    if ip_perm["PrefixListIds"]:
                        port_is_open = True
                    if not port_is_open:
                        log.info(
                            f"The inbound rule does not appear to be open to
either this computer's IP "
                            f"address of {ip_address}, to all IP addresses
(0.0.0.0/0), or to a prefix list ID."
                        )
                    else:
                        break
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to verify inbound rule for port {port} for VPC
{vpc['VpcId']}."
            )
            if error_code == "InvalidVpcID.NotFound":
                log.error(
                    f"The specified VPC ID '{vpc['VpcId']}' does not exist.
Please check the VPC ID."
                )
```

```
            log.error(f"Full error:\n\t{err}")
        else:
            return sec_group, port_is_open


    def open_inbound_port(self, sec_group_id: str, port: int, ip_address: str) ->
None:
        """
        Add an ingress rule to the specified security group that allows access on
the
        specified port from the specified IP address.

        :param sec_group_id: The ID of the security group to modify.
        :param port: The port to open.
        :param ip_address: The IP address that is granted access.
        """
        try:
            self.ec2_client.authorize_security_group_ingress(
                GroupId=sec_group_id,
                CidrIp=f"{ip_address}/32",
                FromPort=port,
                ToPort=port,
                IpProtocol="tcp",
            )
            log.info(
                "Authorized ingress to %s on port %s from %s.",
                sec_group_id,
                port,
                ip_address,
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to authorize ingress to security group '{sec_group_id}'
on port {port} from {ip_address}."
            )
            if error_code == "InvalidGroupId.Malformed":
                log.error(
                    "The security group ID is malformed. "
                    "Please verify that the security group ID is correct."
                )
            elif error_code == "InvalidPermission.Duplicate":
                log.error(
                    "The specified rule already exists in the security group. "
```

```
                    "Check the existing rules for this security group."
                )
            log.error(f"Full error:\n\t{err}")


    def get_subnets(self, vpc_id: str, zones: List[str] = None) -> List[Dict[str,
Any]]:
        """
        Gets the default subnets in a VPC for a specified list of Availability
Zones.

        :param vpc_id: The ID of the VPC to look up.
        :param zones: The list of Availability Zones to look up.
        :return: The list of subnets found.
        """
        # Ensure that 'zones' is a list, even if None is passed
        if zones is None:
            zones = []
        try:
            paginator = self.ec2_client.get_paginator("describe_subnets")
            page_iterator = paginator.paginate(
                Filters=[
                    {"Name": "vpc-id", "Values": [vpc_id]},
                    {"Name": "availability-zone", "Values": zones},
                    {"Name": "default-for-az", "Values": ["true"]},
                ]
            )

            subnets = []
            for page in page_iterator:
                subnets.extend(page["Subnets"])

            log.info("Found %s subnets for the specified zones.", len(subnets))
            return subnets
        except ClientError as err:
            log.error(
                f"Failed to retrieve subnets for VPC '{vpc_id}' in zones
{zones}."
            )
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidVpcID.NotFound":
                log.error(
                    "The specified VPC ID does not exist. "
                    "Please check the VPC ID and try again."
```

```
                )
            # Add more error-specific handling as needed
            log.error(f"Full error:\n\t{err}")
```

Create a class that wraps Elastic Load Balancing actions.

```
class ElasticLoadBalancerWrapper:
    """Encapsulates Elastic Load Balancing (ELB) actions."""

    def __init__(self, elb_client: boto3.client):
        """
        Initializes the LoadBalancer class with the necessary parameters.
        """
        self.elb_client = elb_client


    def create_target_group(
        self, target_group_name: str, protocol: str, port: int, vpc_id: str
    ) -> Dict[str, Any]:
        """
        Creates an Elastic Load Balancing target group. The target group
 specifies how
        the load balancer forwards requests to instances in the group and how
 instance
        health is checked.

        To speed up this demo, the health check is configured with shortened
 times and
        lower thresholds. In production, you might want to decrease the
 sensitivity of
        your health checks to avoid unwanted failures.

        :param target_group_name: The name of the target group to create.
        :param protocol: The protocol to use to forward requests, such as 'HTTP'.
        :param port: The port to use to forward requests, such as 80.
        :param vpc_id: The ID of the VPC in which the load balancer exists.
        :return: Data about the newly created target group.
        """
        try:
```

```
            response = self.elb_client.create_target_group(
                Name=target_group_name,
                Protocol=protocol,
                Port=port,
                HealthCheckPath="/healthcheck",
                HealthCheckIntervalSeconds=10,
                HealthCheckTimeoutSeconds=5,
                HealthyThresholdCount=2,
                UnhealthyThresholdCount=2,
                VpcId=vpc_id,
            )
            target_group = response["TargetGroups"][0]
            log.info(f"Created load balancing target group
'{target_group_name}'.")
            return target_group
        except ClientError as err:
            log.error(
                f"Couldn't create load balancing target group
'{target_group_name}'."
            )
            error_code = err.response["Error"]["Code"]

            if error_code == "DuplicateTargetGroupName":
                log.error(
                    f"Target group name {target_group_name} already exists. "
                    "Check if the target group already exists."
                    "Consider using a different name or deleting the existing
target group if appropriate."
                )
            elif error_code == "TooManyTargetGroups":
                log.error(
                    "Too many target groups exist in the account. "
                    "Consider deleting unused target groups to create space for
new ones."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_target_group(self, target_group_name) -> None:
        """
        Deletes the target group.
        """
        try:
            # Describe the target group to get its ARN
```

```
            response =
self.elb_client.describe_target_groups(Names=[target_group_name])
            tg_arn = response["TargetGroups"][0]["TargetGroupArn"]

            # Delete the target group
            self.elb_client.delete_target_group(TargetGroupArn=tg_arn)
            log.info("Deleted load balancing target group %s.",
target_group_name)

            # Use a custom waiter to wait until the target group is no longer
available
            self.wait_for_target_group_deletion(self.elb_client, tg_arn)
            log.info("Target group %s successfully deleted.", target_group_name)

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to delete target group '{target_group_name}'.")
            if error_code == "TargetGroupNotFound":
                log.error(
                    "Load balancer target group either already deleted or never
existed. "
                    "Verify the name and check that the resource exists in the
AWS Console."
                )
            elif error_code == "ResourceInUseException":
                log.error(
                    "Target group still in use by another resource. "
                    "Ensure that the target group is no longer associated with
any load balancers or resources.",
                )
            log.error(f"Full error:\n\t{err}")

    def wait_for_target_group_deletion(
        self, elb_client, target_group_arn, max_attempts=10, delay=30
    ):
        for attempt in range(max_attempts):
            try:

elb_client.describe_target_groups(TargetGroupArns=[target_group_arn])
                print(
                    f"Attempt {attempt + 1}: Target group {target_group_arn}
still exists."
                )
            except ClientError as e:
```

```
                if e.response["Error"]["Code"] == "TargetGroupNotFound":
                    print(
                        f"Target group {target_group_arn} has been successfully
deleted."
                    )
                    return
                else:
                    raise
            time.sleep(delay)
        raise TimeoutError(
            f"Target group {target_group_arn} was not deleted after {max_attempts
* delay} seconds."
        )


    def create_load_balancer(
        self,
        load_balancer_name: str,
        subnet_ids: List[str],
    ) -> Dict[str, Any]:
        """
        Creates an Elastic Load Balancing load balancer that uses the specified
subnets
        and forwards requests to the specified target group.

        :param load_balancer_name: The name of the load balancer to create.
        :param subnet_ids: A list of subnets to associate with the load balancer.
        :return: Data about the newly created load balancer.
        """
        try:
            response = self.elb_client.create_load_balancer(
                Name=load_balancer_name, Subnets=subnet_ids
            )
            load_balancer = response["LoadBalancers"][0]
            log.info(f"Created load balancer '{load_balancer_name}'.")

            waiter = self.elb_client.get_waiter("load_balancer_available")
            log.info(
                f"Waiting for load balancer '{load_balancer_name}' to be
available..."
            )
            waiter.wait(Names=[load_balancer_name])
            log.info(f"Load balancer '{load_balancer_name}' is now available!")
```

```
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to create load balancer '{load_balancer_name}'. Error
code: {error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "DuplicateLoadBalancerNameException":
                log.error(
                    f"A load balancer with the name '{load_balancer_name}'
already exists. "
                    "Load balancer names must be unique within the AWS region. "
                    "Please choose a different name and try again."
                )
            if error_code == "TooManyLoadBalancersException":
                log.error(
                    "The maximum number of load balancers has been reached in
this account and region. "
                    "You can delete unused load balancers or request an increase
in the service quota from AWS Support."
                )
            log.error(f"Full error:\n\t{err}")
        else:
            return load_balancer


    def create_listener(
        self,
        load_balancer_name: str,
        target_group: Dict[str, Any],
    ) -> Dict[str, Any]:
        """
        Creates a listener for the specified load balancer that forwards requests
to the
        specified target group.

        :param load_balancer_name: The name of the load balancer to create a
listener for.
        :param target_group: An existing target group that is added as a listener
to the
                                load balancer.
        :return: Data about the newly created listener.
        """
        try:
```

```
            # Retrieve the load balancer ARN
            load_balancer_response = self.elb_client.describe_load_balancers(
                Names=[load_balancer_name]
            )
            load_balancer_arn = load_balancer_response["LoadBalancers"][0][
                "LoadBalancerArn"
            ]

            # Create the listener
            response = self.elb_client.create_listener(
                LoadBalancerArn=load_balancer_arn,
                Protocol=target_group["Protocol"],
                Port=target_group["Port"],
                DefaultActions=[
                    {
                        "Type": "forward",
                        "TargetGroupArn": target_group["TargetGroupArn"],
                    }
                ],
            )
            log.info(
                f"Created listener to forward traffic from load balancer
'{load_balancer_name}' to target group '{target_group['TargetGroupName']}'."
            )
            return response["Listeners"][0]
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to add a listener on '{load_balancer_name}' for target
group '{target_group['TargetGroupName']}'."
            )

            if error_code == "ListenerNotFoundException":
                log.error(
                    f"The listener could not be found for the load balancer
'{load_balancer_name}'. "
                    "Please check the load balancer name and target group
configuration."
                )
            if error_code == "InvalidConfigurationRequestException":
                log.error(
                    f"The configuration provided for the listener on load
balancer '{load_balancer_name}' is invalid. "
```

```
                    "Please review the provided protocol, port, and target group
settings."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_load_balancer(self, load_balancer_name) -> None:
        """
        Deletes a load balancer.

        :param load_balancer_name: The name of the load balancer to delete.
        """
        try:
            response = self.elb_client.describe_load_balancers(
                Names=[load_balancer_name]
            )
            lb_arn = response["LoadBalancers"][0]["LoadBalancerArn"]
            self.elb_client.delete_load_balancer(LoadBalancerArn=lb_arn)
            log.info("Deleted load balancer %s.", load_balancer_name)
            waiter = self.elb_client.get_waiter("load_balancers_deleted")
            log.info("Waiting for load balancer to be deleted...")
            waiter.wait(Names=[load_balancer_name])
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Couldn't delete load balancer '{load_balancer_name}'. Error
code: {error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "LoadBalancerNotFoundException":
                log.error(
                    f"The load balancer '{load_balancer_name}' does not exist. "
                    "Please check the name and try again."
                )
            log.error(f"Full error:\n\t{err}")


    def get_endpoint(self, load_balancer_name) -> str:
        """
        Gets the HTTP endpoint of the load balancer.

        :return: The endpoint.
        """
        try:
```

```
            response = self.elb_client.describe_load_balancers(
                Names=[load_balancer_name]
            )
            return response["LoadBalancers"][0]["DNSName"]
        except ClientError as err:
            log.error(
                f"Couldn't get the endpoint for load balancer
{load_balancer_name}"
            )
            error_code = err.response["Error"]["Code"]
            if error_code == "LoadBalancerNotFoundException":
                log.error(
                    "Verify load balancer name and ensure it exists in the AWS
console."
                )
            log.error(f"Full error:\n\t{err}")

    @staticmethod
    def verify_load_balancer_endpoint(endpoint) -> bool:
        """
        Verify this computer can successfully send a GET request to the load
balancer endpoint.

        :param endpoint: The endpoint to verify.
        :return: True if the GET request is successful, False otherwise.
        """
        retries = 3
        verified = False
        while not verified and retries > 0:
            try:
                lb_response = requests.get(f"http://{endpoint}")
                log.info(
                    "Got response %s from load balancer endpoint.",
                    lb_response.status_code,
                )
                if lb_response.status_code == 200:
                    verified = True
                else:
                    retries = 0
            except requests.exceptions.ConnectionError:
                log.info(
                    "Got connection error from load balancer endpoint,
retrying..."
                )
```

```
                retries -= 1
                time.sleep(10)
        return verified

    def check_target_health(self, target_group_name: str) -> List[Dict[str,
Any]]:
        """
        Checks the health of the instances in the target group.

        :return: The health status of the target group.
        """
        try:
            tg_response = self.elb_client.describe_target_groups(
                Names=[target_group_name]
            )
            health_response = self.elb_client.describe_target_health(
                TargetGroupArn=tg_response["TargetGroups"][0]["TargetGroupArn"]
            )
        except ClientError as err:
            log.error(f"Couldn't check health of {target_group_name} target(s).")
            error_code = err.response["Error"]["Code"]
            if error_code == "LoadBalancerNotFoundException":
                log.error(
                    "Load balancer associated with the target group was not
found. "
                    "Ensure the load balancer exists, is in the correct AWS
region, and "
                    "that you have the necessary permissions to access it.",
                )
            elif error_code == "TargetGroupNotFoundException":
                log.error(
                    "Target group was not found. "
                    "Verify the target group name, check that it exists in the
correct region, "
                    "and ensure it has not been deleted or created in a different
account.",
                )
            log.error(f"Full error:\n\t{err}")
        else:
            return health_response["TargetHealthDescriptions"]
```

Create a class that uses DynamoDB to simulate a recommendation service.

```
class RecommendationService:
    """
    Encapsulates a DynamoDB table to use as a service that recommends books,
 movies,
    and songs.
    """

    def __init__(self, table_name: str, dynamodb_client: boto3.client):
        """
        Initializes the RecommendationService class with the necessary
 parameters.

        :param table_name: The name of the DynamoDB recommendations table.
        :param dynamodb_client: A Boto3 DynamoDB client.
        """
        self.table_name = table_name
        self.dynamodb_client = dynamodb_client

    def create(self) -> Dict[str, Any]:
        """
        Creates a DynamoDB table to use as a recommendation service. The table
 has a
        hash key named 'MediaType' that defines the type of media recommended,
 such as
        Book or Movie, and a range key named 'ItemId' that, combined with the
 MediaType,
        forms a unique identifier for the recommended item.

        :return: Data about the newly created table.
        :raises RecommendationServiceError: If the table creation fails.
        """
        try:
            response = self.dynamodb_client.create_table(
                TableName=self.table_name,
                AttributeDefinitions=[
                    {"AttributeName": "MediaType", "AttributeType": "S"},
                    {"AttributeName": "ItemId", "AttributeType": "N"},
                ],
                KeySchema=[
```

```
                    {"AttributeName": "MediaType", "KeyType": "HASH"},
                    {"AttributeName": "ItemId", "KeyType": "RANGE"},
                ],
                ProvisionedThroughput={"ReadCapacityUnits": 5,
"WriteCapacityUnits": 5},
            )
            log.info("Creating table %s...", self.table_name)
            waiter = self.dynamodb_client.get_waiter("table_exists")
            waiter.wait(TableName=self.table_name)
            log.info("Table %s created.", self.table_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceInUseException":
                log.info("Table %s exists, nothing to be done.", self.table_name)
            else:
                raise RecommendationServiceError(
                    self.table_name, f"ClientError when creating table: {err}."
                )
        else:
            return response

    def populate(self, data_file: str) -> None:
        """
        Populates the recommendations table from a JSON file.

        :param data_file: The path to the data file.
        :raises RecommendationServiceError: If the table population fails.
        """
        try:
            with open(data_file) as data:
                items = json.load(data)
            batch = [{"PutRequest": {"Item": item}} for item in items]
            self.dynamodb_client.batch_write_item(RequestItems={self.table_name:
batch})
            log.info(
                "Populated table %s with items from %s.", self.table_name,
data_file
            )
        except ClientError as err:
            raise RecommendationServiceError(
                self.table_name, f"Couldn't populate table from {data_file}:
{err}"
            )

    def destroy(self) -> None:
```

```
        """
        Deletes the recommendations table.

        :raises RecommendationServiceError: If the table deletion fails.
        """
        try:
            self.dynamodb_client.delete_table(TableName=self.table_name)
            log.info("Deleting table %s...", self.table_name)
            waiter = self.dynamodb_client.get_waiter("table_not_exists")
            waiter.wait(TableName=self.table_name)
            log.info("Table %s deleted.", self.table_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                log.info("Table %s does not exist, nothing to do.",
 self.table_name)
            else:
                raise RecommendationServiceError(
                    self.table_name, f"ClientError when deleting table: {err}."
                )
```

Create a class that wraps Systems Manager actions.

```
class ParameterHelper:
    """
    Encapsulates Systems Manager parameters. This example uses these parameters
 to drive
    the demonstration of resilient architecture, such as failure of a dependency
 or
    how the service responds to a health check.
    """

    table: str = "doc-example-resilient-architecture-table"
    failure_response: str = "doc-example-resilient-architecture-failure-response"
    health_check: str = "doc-example-resilient-architecture-health-check"

    def __init__(self, table_name: str, ssm_client: boto3.client):
        """
        Initializes the ParameterHelper class with the necessary parameters.
```

```
        :param table_name: The name of the DynamoDB table that is used as a
recommendation
                           service.
        :param ssm_client: A Boto3 Systems Manager client.
        """
        self.ssm_client = ssm_client
        self.table_name = table_name

    def reset(self) -> None:
        """
        Resets the Systems Manager parameters to starting values for the demo.
        These are the name of the DynamoDB recommendation table, no response when
a
        dependency fails, and shallow health checks.
        """
        self.put(self.table, self.table_name)
        self.put(self.failure_response, "none")
        self.put(self.health_check, "shallow")

    def put(self, name: str, value: str) -> None:
        """
        Sets the value of a named Systems Manager parameter.

        :param name: The name of the parameter.
        :param value: The new value of the parameter.
        :raises ParameterHelperError: If the parameter value cannot be set.
        """
        try:
            self.ssm_client.put_parameter(
                Name=name, Value=value, Overwrite=True, Type="String"
            )
            log.info("Setting parameter %s to '%s'.", name, value)
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to set parameter {name}.")
            if error_code == "ParameterLimitExceeded":
                log.error(
                    "The parameter limit has been exceeded. "
                    "Consider deleting unused parameters or request a limit
increase."
                )
            elif error_code == "ParameterAlreadyExists":
                log.error(
```

```
                    "The parameter already exists and overwrite is set to False.
"
                    "Use Overwrite=True to update the parameter."
                )
            log.error(f"Full error:\n\t{err}")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation

- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Troubleshoot issues in Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling provides specific and descriptive errors to help you troubleshoot issues. You can find the error messages in the description of the scaling activities.

Topics

- Retrieve an error message from scaling activities
- Turn off scaling activities
- Additional troubleshooting resources
- Troubleshoot Amazon EC2 Auto Scaling: EC2 instance launch failures
- Troubleshoot Amazon EC2 Auto Scaling: AMI issues
- Troubleshoot Amazon EC2 Auto Scaling: Load balancer issues
- Troubleshoot Amazon EC2 Auto Scaling: Launch templates

Retrieve an error message from scaling activities

To retrieve an error message from the description of scaling activities, use the describe-scaling-activities command. You have a record of scaling activities that dates back 6 weeks. Scaling activities are ordered by start time, with the latest scaling activities listed first.

> **Note**
>
> The scaling activities are also displayed in the activity history in the Amazon EC2 Auto Scaling console on the **Activity** tab for the Auto Scaling group.

To see the scaling activities for a specific Auto Scaling group, use the following command.

```
aws autoscaling describe-scaling-activities --auto-scaling-group-name my-asg
```

The following is an example response, where `StatusCode` contains the current status of the activity and `StatusMessage` contains the error message.

```
{
    "Activities": [
```

```
        {
            "ActivityId": "3b05dbf6-037c-b92f-133f-38275269dc0f",
            "AutoScalingGroupName": "my-asg",
            "Description": "Launching a new EC2 instance: i-003a5b3ffe1e9358e.  Status
 Reason: Instance failed to complete user's Lifecycle Action: Lifecycle Action with
 token e85eb647-4fe0-4909-b341-a6c42d8aba1f was abandoned: Lifecycle Action Completed
 with ABANDON Result",
            "Cause": "At 2021-01-11T00:35:52Z a user request created an
 AutoScalingGroup changing the desired capacity from 0 to 1.  At 2021-01-11T00:35:53Z
 an instance was started in response to a difference between desired and actual
 capacity, increasing the capacity from 0 to 1.",
            "StartTime": "2021-01-11T00:35:55.542Z",
            "EndTime": "2021-01-11T01:06:31Z",
            "StatusCode": "Cancelled",
            "StatusMessage": "Instance failed to complete user's Lifecycle Action:
 Lifecycle Action with token e85eb647-4fe0-4909-b341-a6c42d8aba1f was abandoned:
 Lifecycle Action Completed with ABANDON Result",
            "Progress": 100,
            "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":\"us-
west-2b\"...}",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:283179a2-
f3ce-423d-93f6-66bb518232f7:autoScalingGroupName/my-asg"
        },
     ...
    ]
}
```

For a description of the fields in the output, see Activity in the *Amazon EC2 Auto Scaling API Reference*.

To view scaling activities for a deleted group

To view scaling activities after the Auto Scaling group has been deleted, add the `--include-deleted-groups` option to the describe-scaling-activities command as follows.

```
aws autoscaling describe-scaling-activities --auto-scaling-group-name my-asg --include-
deleted-groups
```

The following is an example response, with a scaling activity for a deleted group.

```
{
    "Activities": [
```

```
        {
            "ActivityId": "e1f5de0e-f93e-1417-34ac-092a76fba220",
            "AutoScalingGroupName": "my-asg",
            "Description": "Launching a new EC2 instance.  Status Reason: Your Spot
 request price of 0.001 is lower than the minimum required Spot request fulfillment
 price of 0.0031. Launching EC2 instance failed.",
            "Cause": "At 2021-01-13T20:47:24Z a user request update of AutoScalingGroup
 constraints to min: 1, max: 5, desired: 3 changing the desired capacity from 0 to 3.
  At 2021-01-13T20:47:27Z an instance was started in response to a difference between
 desired and actual capacity, increasing the capacity from 0 to 3.",
            "StartTime": "2021-01-13T20:47:30.094Z",
            "EndTime": "2021-01-13T20:47:30Z",
            "StatusCode": "Failed",
            "StatusMessage": "Your Spot request price of 0.001 is lower than the
 minimum required Spot request fulfillment price of 0.0031. Launching EC2 instance
 failed.",
            "Progress": 100,
            "Details": "{\"Subnet ID\":\"subnet-5ea0c127\",\"Availability Zone\":\"us-
west-2b\"...}",
            "AutoScalingGroupState": "Deleted",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:123456789012:autoScalingGroup:283179a2-
f3ce-423d-93f6-66bb518232f7:autoScalingGroupName/my-asg"
        },
     ...
    ]
}
```

Turn off scaling activities

You have the following options if you need to investigate an issue without interference from scaling policies or scheduled actions:

- Prevent all dynamic scaling policies and scheduled actions from making changes to the group's desired capacity by suspending the `AlarmNotification` and `ScheduledActions` processes. For more information, see Suspend and resume Amazon EC2 Auto Scaling processes.
- Disable individual dynamic scaling policies so that they don't change the group's desired capacity in response to changes in load. For more information, see Disable a scaling policy for an Auto Scaling group.
- Update individual target tracking scaling policies to only scale out (add capacity) by disabling the policy's scale-in portion. This method prevents the group's desired capacity from shrinking but

allows it to be increased when load increases. For more information, see Target tracking scaling policies for Amazon EC2 Auto Scaling.

- Update your predictive scaling policy to *forecast only* mode. While in forecast only mode, predictive scaling will continue to generate forecasts, but it will not automatically increase capacity. For more information, see Create a predictive scaling policy for an Auto Scaling group.

Additional troubleshooting resources

The following pages provide additional information for troubleshooting issues with Amazon EC2 Auto Scaling.

- Verify a scaling activity for an Auto Scaling group
- View monitoring graphs in the Amazon EC2 Auto Scaling console
- Health checks for instances in an Auto Scaling group
- Considerations and limitations for lifecycle hooks
- Complete a lifecycle action in an Auto Scaling group
- Provide network connectivity for your Auto Scaling instances using Amazon VPC
- Temporarily remove instances from your Auto Scaling group
- Disable a scaling policy for an Auto Scaling group
- Suspend and resume Amazon EC2 Auto Scaling processes
- Control which Auto Scaling instances terminate during scale in
- Delete your Auto Scaling infrastructure
- Quotas for Auto Scaling resources and groups

The following AWS resources can also be of help:

- Amazon EC2 Auto Scaling topics in the AWS Knowledge Center
- Amazon EC2 Auto Scaling questions on AWS re:Post
- Amazon EC2 Auto Scaling posts in the AWS Compute Blog
- Troubleshooting CloudFormation in the *AWS CloudFormation User Guide*

Troubleshooting often requires iterative query and discovery by an expert or from a community of helpers. If you continue to experience issues after trying the suggestions in this section, contact

AWS Support (in the AWS Management Console, click **Support**, **Support Center**) or ask a question on AWS re:Post using the **Amazon EC2 Auto Scaling** tag.

Troubleshoot Amazon EC2 Auto Scaling: EC2 instance launch failures

This page provides information about your EC2 instances that fail to launch, potential causes, and the steps you can take to resolve the issues.

To retrieve an error message, see Retrieve an error message from scaling activities.

When your EC2 instances fail to launch, you might get one or more of the following error messages:

Launch issues

- The requested configuration is currently not supported.
- The security group <name of the security group> does not exist. Launching EC2 instance failed.
- The key pair <key pair associated with your EC2 instance> does not exist. Launching EC2 instance failed.
- Your requested instance type (<instance type>) is not supported in your requested Availability Zone (<instance Availability Zone>)...
- Your Spot request price of 0.015 is lower than the minimum required Spot request fulfillment price of 0.0735...
- Invalid device name <device name> / Invalid device name upload. Launching EC2 instance failed.
- Value (<name associated with the instance storage device>) for parameter virtualName is invalid... Launching EC2 instance failed.
- EBS block device mappings not supported for instance-store AMIs.
- Placement groups may not be used with instances of type '<instance type>'. Launching EC2 instance failed.
- Client.InternalError: Client error on launch.
- We currently do not have sufficient <instance type> capacity in the Availability Zone you requested... Launching EC2 instance failed.
- The requested reservation does not have sufficient compatible and available capacity for this request. Launching EC2 instance failed.
- Your Capacity Block reservation <reservation id> is not active yet. Launching EC2 instance failed.

- There is no Spot capacity available that matches your request. Launching EC2 instance failed.
- <number of instances> instance(s) are already running. Launching EC2 instance failed.

The requested configuration is currently not supported.

Cause: Some options in your launch template or launch configuration might not be compatible with the instance type, or the instance configuration might not be supported in your requested AWS Region or Availability Zones.

Solution: Try a different instance configuration. To search for an instance type that meets your requirements, see Finding an Amazon EC2 instance type in the *Amazon EC2 User Guide*.

For further guidance to resolve this issue, check the following:

- Ensure that you have chosen an AMI that is supported by your instance type. For example, if the instance type uses an Arm-based AWS Graviton processor instead of an Intel Xeon processor, you need an Arm-compatible AMI. For more information about choosing a compatible instance type, see Compatibility for changing the instance type in the *Amazon EC2 User Guide*.
- Test that the instance type is available in your requested Region and Availability Zones. The newest generation instance types might not yet be available in a given Region or Availability Zone. The older generation instance types might not be available in newer Regions and Availability Zones. To search for instance types offered by location (Region or Availability Zone), use the describe-instance-type-offerings command. For more information, see Finding an Amazon EC2 instance type in the *Amazon EC2 User Guide*.
- If you use Dedicated Instances or Dedicated Hosts, ensure that you have chosen an instance type that's supported as a Dedicated Instance or Dedicated Host.

The security group <name of the security group> does not exist. Launching EC2 instance failed.

Cause: The security group specified in your launch template or launch configuration might have been deleted.

Solution:

1. Use the describe-security-groups command to get the list of the security groups associated with your account.

2. From the list, select the security groups to use. To create a security group instead, use the create-security-group command.
3. Create a new launch template or launch configuration.
4. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

The key pair <key pair associated with your EC2 instance> does not exist. Launching EC2 instance failed.

Cause: The key pair that was used when launching the instance might have been deleted.

Solution:

1. Use the describe-key-pairs command to get the list of the key pairs available to you.
2. From the list, select the key pair to use. To create a key pair instead, use the create-key-pair command.
3. Create a new launch template or launch configuration.
4. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

Your requested instance type (<instance type>) is not supported in your requested Availability Zone (<instance Availability Zone>)...

Error message: Your requested instance type (<instance type>) is not supported in your requested Availability Zone (<instance Availability Zone>)...Launching EC2 instance failed.

Cause: The Availability Zones specified in your Auto Scaling group don't support your chosen instance type.

Solution:

1. Verify which Availability Zones support your chosen instance type using the describe-instance-type-offerings command or from the Amazon EC2 console by checking the **Availability Zones** value on the networking pane of the **Instance types** page.

2. Update or remove the subnet for any unsupported zones in the settings of your Auto Scaling group using the update-auto-scaling-group command. For more information, see Add an Availability Zone.

Your Spot request price of 0.015 is lower than the minimum required Spot request fulfillment price of 0.0735...

Cause: The Spot maximum price in your request is lower than the Spot price for the instance type that you selected.

Solution: Submit a new request with a higher Spot maximum price (possibly the On-Demand price). Previously, the Spot price you paid was based on bidding. Today, you pay the current Spot price. By setting your maximum price higher, it gives the Amazon EC2 Spot service a better chance of launching and maintaining your required amount of capacity.

Invalid device name <device name> / Invalid device name upload. Launching EC2 instance failed.

Cause 1: The block device mappings in your launch template or launch configuration might contain block device names that are not available or currently not supported.

Solution:

1. Verify which device names are available for your specific instance configuration. For more details on device naming, see Device names on Linux instances in the *Amazon EC2 User Guide*.
2. Manually create an Amazon EC2 instance that is not part of the Auto Scaling group and investigate the problem. If the block device naming configuration conflicts with the names in the Amazon Machine Image (AMI), the instance will fail during launch. For more information, see Block device mappings in the *Amazon EC2 User Guide*.
3. After you confirm that your instance launched successfully, use the describe-volumes command to see how the volumes are exposed to the instance.
4. Create a new launch template or launch configuration using the device name listed in the volume description.
5. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

Value (<name associated with the instance storage device>) for parameter virtualName is invalid... Launching EC2 instance failed.

Cause: The format specified for the virtual name associated with the block device is incorrect.

Solution:

1. Create a new launch template or launch configuration by specifying the device name in the `virtualName` parameter. For information about the device name format, see Device naming on Linux instances in the *Amazon EC2 User Guide*.
2. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

EBS block device mappings not supported for instance-store AMIs.

Cause: The block device mappings specified in the launch template or launch configuration are not supported on your instance.

Solution:

1. Create a new launch template or launch configuration with block device mappings supported by your instance type. For more information, see Block device mapping in the *Amazon EC2 User Guide*.
2. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

Placement groups may not be used with instances of type '<instance type>'. Launching EC2 instance failed.

Cause: Your cluster placement group contains an invalid instance type.

Solution:

1. For information about valid instance types supported by the placement groups, see Placement groups in the *Amazon EC2 User Guide*.
2. Follow the instructions detailed in the Placement groups to create a new placement group.

3. Alternatively, create a new launch template or launch configuration with the supported instance type.
4. Update your Auto Scaling group with a new placement group, launch template, or launch configuration using the update-auto-scaling-group command.

Client.InternalError: Client error on launch.

Problem: Amazon EC2 Auto Scaling tries to launch an instance that has an encrypted EBS volume, but the service-linked role does not have access to the AWS KMS customer managed key used to encrypt it. For more information, see Required AWS KMS key policy for use with encrypted volumes.

Cause 1: You need a key policy that gives permission to use the customer managed key to the proper service-linked role.

Solution 1: Allow the service-linked role to use the customer managed key as follows:

1. Determine which service-linked role to use for this Auto Scaling group.
2. Update the key policy on the customer managed key and allow the service-linked role to use the customer managed key.
3. Update the Auto Scaling group to use the service-linked role.

For an example of a key policy that lets the service-linked role use the customer managed key, see Example 1: Key policy sections that allow access to the customer managed key.

Cause 2: If the customer managed key and Auto Scaling group are in *different* AWS accounts, you need to configure cross-account access to the customer managed key in order to give permission to use the customer managed key to the proper service-linked role.

Solution 2: Allow the service-linked role in the external account to use the customer managed key in the local account as follows:

1. Update the key policy on the customer managed key to allow the Auto Scaling group account access to the customer managed key.
2. Define an IAM user or role in the Auto Scaling group account that can create a grant.
3. Determine which service-linked role to use for this Auto Scaling group.

4. Create a grant to the customer managed key with the proper service-linked role as the grantee principal.
5. Update the Auto Scaling group to use the service-linked role.

For more information, see Example 2: Key policy sections that allow cross-account access to the customer managed key.

Solution 3: Use a customer managed key in the same AWS account as the Auto Scaling group.

1. Copy and re-encrypt the snapshot with another customer managed key that belongs to the same account as the Auto Scaling group.
2. Allow the service-linked role to use the new customer managed key. See the steps for Solution 1.

We currently do not have sufficient <instance type> capacity in the Availability Zone you requested... Launching EC2 instance failed.

Error message: We currently do not have sufficient <instance type> capacity in the Availability Zone you requested (<requested Availability Zone>). Our system will be working on provisioning additional capacity. You can currently get <instance type> capacity by not specifying an Availability Zone in your request or choosing <list of Availability Zones that currently supports the instance type>. Launching EC2 instance failed.

Cause: At this time, the requested instance type and Availability Zone combination isn't supported.

Solution: To resolve the issue, try the following:

- Wait a few minutes for Amazon EC2 Auto Scaling to find capacity for this instance type in other enabled Availability Zones.
- Expand your Auto Scaling group to additional Availability Zones. For more information, see Add an Availability Zone.
- Follow the best practice of using a diverse set of instance types so that you're not reliant on one specific instance type. For more information, see Auto Scaling groups with multiple instance types and purchase options.

The requested reservation does not have sufficient compatible and available capacity for this request. Launching EC2 instance failed.

Cause 1: You've reached the limit on the number of instances that you can launch with a `targeted` On-Demand Capacity Reservation.

Solution 1: Either increase the number of instances that you can launch with the `targeted` On-Demand Capacity Reservation, or use a Capacity Reservations group so that anything beyond the reserved capacity will launch as regular On-Demand capacity. For more information, see Reserve capacity in specific Availability Zones with Capacity Reservations .

Cause 2: You've reached the limit on the number of instances that you can launch with a Capacity Block.

With Capacity Blocks, you are constrained by the amount of capacity originally purchased. If you experience a higher number of launches than anticipated and use up all the capacity that you have available, this causes launches to fail. Terminating instances go through a lengthy clean up process before they're fully terminated. During this time, they can't be reused. This can also cause launches to fail. For more information, see Use Capacity Blocks for machine learning workloads.

Solution 2: To resolve the issue, try the following:

- Keep the request as is. If a Capacity Block instance is terminating, you must wait several minutes for the instance to finish terminating and capacity to become available again. Amazon EC2 Auto Scaling continues to automatically make the launch request until capacity becomes available.
- Make sure that you purchase sufficient capacity to accommodate your peak workload so that you do not encounter this error frequently.

Your Capacity Block reservation <reservation id> is not active yet. Launching EC2 instance failed.

Cause: The specified Capacity Block is not active yet.

Solution: Follow the recommended approach for Capacity Blocks and use scheduled scaling. Doing so helps you make sure you increase the desired capacity of your Auto Scaling group only when the reservation is active and decrease it before the reservation is over.

There is no Spot capacity available that matches your request. Launching EC2 instance failed.

Cause: At this time, there isn't enough spare capacity to fulfill your request for Spot Instances.

Solution: To resolve the issue, try the following:

- Wait a few minutes; capacity can shift frequently. Amazon EC2 Auto Scaling continues to automatically make the launch request until capacity becomes available.
- Expand your Auto Scaling group to additional Availability Zones. For more information, see Add an Availability Zone.
- Follow the best practice of using a diverse set of instance types so that you're not reliant on one specific instance type. For more information, see Auto Scaling groups with multiple instance types and purchase options.

<number of instances> instance(s) are already running. Launching EC2 instance failed.

Cause: You have reached the limit on the number of instances that you can launch in a Region. When you create your AWS account, we set default limits on the number of instances you can run on a per-Region basis.

Solution: To resolve the issue, try the following:

- If your current limits aren't adequate for your needs, you can request a quota increase on a per-Region basis. For more information, see Amazon EC2 service quotas in the *Amazon EC2 User Guide*.
- Submit a new request with a reduced number of instances (which you can increase at a later stage).

Troubleshoot Amazon EC2 Auto Scaling: AMI issues

This page provides information about the issues associated with your AMIs, potential causes, and the steps you can take to resolve the issues.

To retrieve an error message, see Retrieve an error message from scaling activities.

When your EC2 instances fail to launch due to issues with your AMI, you might get one or more of the following error messages.

AMI issues

- The AMI ID <ID of your AMI> does not exist. Launching EC2 instance failed.
- AMI <AMI ID> is pending, and cannot be run. Launching EC2 instance failed.
- Invalid device name <device name>. Launching EC2 instance failed.
- The architecture 'arm64 ' of the specified instance type does not match the architecture 'x86_64' of the specified AMI...Launching EC2 instance failed.
- AMI '<AMI ID>' is disabled, and cannot be run. Launching EC2 instance failed.

> **Important**
>
> AWS supports sharing an AMI privately with another AWS account by modifying the AMI permissions. If an AMI is made private without being shared, this can result in an authorization error when launching new instances. For more information about sharing private AMIs, see Share an AMI with specific AWS accounts in the *Amazon EC2 User Guide*.

The AMI ID <ID of your AMI> does not exist. Launching EC2 instance failed.

- **Cause**: The AMI might have been deleted after creating the launch template or launch configuration.
- **Solution**:
  1. Create a new launch template or launch configuration using a valid AMI.
  2. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

AMI <AMI ID> is pending, and cannot be run. Launching EC2 instance failed.

Cause: You might have just created your AMI (by taking a snapshot of a running instance or any other way), and it might not be available yet.

Solution: You must wait for your AMI to be available and then create your launch template or launch configuration.

Invalid device name <device name>. Launching EC2 instance failed.

Cause: When attaching an EBS volume to an EC2 instance, you must provide a valid device name for the volume. The selected AMI must support this device name.

Solution:

1. Create a new launch template or launch configuration and specify the correct device name for your AMI. The recommended naming convention varies based on the virtualization type of the AMI. For more information, see Device names in the *Amazon EC2 User Guide*.
2. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

The architecture 'arm64 ' of the specified instance type does not match the architecture 'x86_64' of the specified AMI...Launching EC2 instance failed.

Cause 1: If the architecture of the AMI and the instance type used in your launch template or launch configuration are not the same, you get an error when Amazon EC2 Auto Scaling tries to launch an instance using the incompatible instance configuration.

Solution 1:

1. Verify the architecture of your AMI using the describe-images command or from the Amazon EC2 console by checking the **Architecture** value on the details pane of the **Amazon Machine Images (AMIs)** page.
2. Find an instance type that has the same architecture as your AMI using the describe-instance-types command or from the Amazon EC2 console by checking the **Architecture** column on the **Instance types** screen. For more information about choosing a compatible instance type, see Compatibility for changing the instance type in the *Amazon EC2 User Guide*.
3. Create a new launch template or launch configuration using an instance type that has the same architecture as your AMI.
4. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

Cause 2: Amazon EC2 Auto Scaling tries to launch an instance type that's specified in the mixed instances policy for your Auto Scaling group, but the instance type does not have the same architecture as the AMI specified in your launch template.

Solution 1: Do not include instance types that have different architectures in your mixed instances policy.

1. Verify the architecture of your AMI using the describe-images command or from the Amazon EC2 console by checking the **Architecture** value on the details pane of the **Amazon Machine Images (AMIs)** page.
2. Verify the architecture of each instance type that you intend to include in your mixed instances policy using the describe-instance-types command or from the Amazon EC2 console by checking the **Architecture** column on the **Instance types** screen. For more information about choosing compatible instance types, see Compatibility for changing the instance type in the *Amazon EC2 User Guide*.
3. Update or remove the incompatible instance types from your Auto Scaling group using the update-auto-scaling-group command.

Solution 2: To launch both Arm (Graviton2) and x86_64 (Intel) instances in the same Auto Scaling group, you must use launch templates supported by an Arm-compatible AMI and an Intel x86-compatible AMI, respectively, to match the instance types in your mixed instances policy.

1. Verify the architecture of the AMI in your existing launch template using the describe-images command or from the Amazon EC2 console by checking the **Architecture** value on the details pane of the **Amazon Machine Images (AMIs)** page.
2. Create a new launch template using an AMI that matches the other architecture you intend to use.
3. Update your Auto Scaling group to override the existing launch template and specify the new launch template for each compatible instance type using the update-auto-scaling-group command. For more information, see Use a different launch template for an instance type.

AMI '<AMI ID>' is disabled, and cannot be run. Launching EC2 instance failed.

Cause: You are attempting to launch instances from an AMI that has been disabled. For more information, see Disable an AMI in the *Amazon EC2 User Guide*.

Solution:

1. Create a new launch template or launch configuration and specify an AMI that is not disabled.
2. Update your Auto Scaling group with the new launch template or launch configuration using the update-auto-scaling-group command.

Troubleshoot Amazon EC2 Auto Scaling: Load balancer issues

This page provides information about issues caused by the load balancer associated with your Auto Scaling group, potential causes, and the steps you can take to resolve the issues.

To retrieve an error message, see Retrieve an error message from scaling activities.

When your EC2 instances fail to launch due to issues with the load balancer associated with your Auto Scaling group, you might get one or more of the following error messages.

Load balancer issues

- One or more target groups not found. Validating load balancer configuration failed.
- Cannot find Load Balancer <your load balancer>. Validating load balancer configuration failed.
- There is no ACTIVE Load Balancer named <load balancer name>. Updating load balancer configuration failed.
- EC2 instance <instance ID> is not in VPC. Updating load balancer configuration failed.

> **Note**
>
> You can use Reachability Analyzer to troubleshoot connectivity issues by checking whether instances in your Auto Scaling group are reachable through the load balancer. To learn about the different network misconfiguration issues that are automatically detected by Reachability Analyzer, see Reachability Analyzer explanation codes in the *Reachability Analyzer User Guide*.

One or more target groups not found. Validating load balancer configuration failed.

Problem: When your Auto Scaling group launches instances, Amazon EC2 Auto Scaling tries to validate that the Elastic Load Balancing resources that are associated with the Auto Scaling group exist. When a target group cannot be found, the scaling activity fails, and you get the `One or more target groups not found. Validating load balancer configuration failed.` error.

Cause 1: A target group attached to your Auto Scaling group has been deleted.

Solution 1: You can either create a new Auto Scaling group without the target group or remove the unused target group from the Auto Scaling group by using the Amazon EC2 Auto Scaling console or the detach-load-balancer-target-groups command.

Cause 2: The target group exists, but there was an issue trying to specify the target group ARN when creating the Auto Scaling group. Resources are not created in the right order.

Solution 2: Create a new Auto Scaling group and specify the target group at the end.

Cannot find Load Balancer <your load balancer>. Validating load balancer configuration failed.

Problem: When your Auto Scaling group launches instances, Amazon EC2 Auto Scaling tries to validate that the Elastic Load Balancing resources that are associated with the Auto Scaling group exist. When a Classic Load Balancer cannot be found, the scaling activity fails, and you get the `Cannot find Load Balancer <your load balancer>. Validating load balancer configuration failed.` error.

Cause 1: The Classic Load Balancer has been deleted.

Solution 1: You can either create a new Auto Scaling group without the load balancer or remove the unused load balancer from the Auto Scaling group by using the Amazon EC2 Auto Scaling console or the detach-load-balancers command.

Cause 2: The Classic Load Balancer exists, but there was an issue trying to specify the load balancer name when creating the Auto Scaling group. Resources are not created in the right order.

Solution 2: Create a new Auto Scaling group and specify the load balancer name at the end.

There is no ACTIVE Load Balancer named <load balancer name>. Updating load balancer configuration failed.

Cause: The specified load balancer might have been deleted.

Solution: You can either create a new load balancer and then create a new Auto Scaling group or create a new Auto Scaling group without the load balancer.

EC2 instance <instance ID> is not in VPC. Updating load balancer configuration failed.

Cause: The specified instance does not exist in the VPC.

Solution: You can either delete your load balancer associated with the instance or create a new Auto Scaling group.

Troubleshoot Amazon EC2 Auto Scaling: Launch templates

Use the following information to help you diagnose and fix common issues that you might encounter when trying to specify a launch template for your Auto Scaling group.

Can't launch instances

If you are unable to launch any instances with an already specified launch template, check the following for general troubleshooting: Troubleshoot Amazon EC2 Auto Scaling: EC2 instance launch failures.

You must use a valid fully-formed launch template (invalid value)

Problem: When you try to specify a launch template for an Auto Scaling group, you get the You `must use a valid fully-formed launch template` error. You might encounter this error because the values in the launch template are only validated when an Auto Scaling group that is using the launch template is created or updated.

Cause 1: If you receive a `You must use a valid fully-formed launch template` error, then there are issues that cause Amazon EC2 Auto Scaling to consider something about the launch template not valid. This is a generic error that can have several different causes.

Solution 1: Try the following steps to troubleshoot:

1. Pay attention to the second part of the error message to find more information. Following the `You must use a valid fully-formed launch template` error, see the more specific error message that identifies the issue that you will need to address.
2. If you are unable to find the cause, test your launch template with the run-instances command. Use the `--dry-run` option, as shown in the following example. This lets you reproduce the issue and can provide insights about its cause.

```
aws ec2 run-instances --launch-template LaunchTemplateName=my-template,Version='1' --
dry-run
```

3. If a value is not valid, verify that the specified resource exists and that it's correct. For example, when you specify an Amazon EC2 key pair, the resource must exist in your account and in the Region in which you are creating or updating your Auto Scaling group.
4. If expected information is missing, verify your settings and adjust the launch template as needed.
5. After making your changes, re-run the run-instances command with the `--dry-run` option to verify that your launch template uses valid values.

For more information, see Create a launch template for an Auto Scaling group.

You are not authorized to use launch template (insufficient permissions)

Problem: When you try to specify a launch template for an Auto Scaling group, you get the `You are not authorized to use launch template` error.

Cause 1: If you are attempting to use a launch template, and the IAM credentials that you are using do not have sufficient permissions, you receive an error that you're not authorized to use the launch template.

Solution 1: To resolve the issue, try the following:

- Verify that the IAM credentials that you are using to make the request has permissions to call the EC2 API actions you need, including the `ec2:RunInstances` action. If you specified any tags in your launch template, you must also have permission to use the `ec2:CreateTags` action.
- Alternatively, verify that the IAM credentials that you are using to make the request is assigned the `AmazonEC2FullAccess` policy. This AWS managed policy grants full access to all Amazon

EC2 resources and related services, including Amazon EC2 Auto Scaling, CloudWatch, and Elastic Load Balancing.

For more information about the permissions required to use launch templates, including example IAM policies, see Control access to launch templates with IAM permissions in the *Amazon EC2 User Guide*. For other example IAM policies, see Control Amazon EC2 launch template usage in Auto Scaling groups.

Cause 2: If you are attempting to use a launch template that specifies an instance profile, you must have IAM permission to pass the IAM role that is associated with the instance profile.

Solution 2: Verify that the IAM credentials that you are using to make the request has the correct `iam:PassRole` permission to pass the specified role to the Amazon EC2 Auto Scaling service. For more information and an example IAM policy, see IAM role for applications that run on Amazon EC2 instances. For further troubleshooting topics related to instance profiles, see Troubleshooting Amazon EC2 and IAM in the *IAM User Guide*.

Cause 3: If you are attempting to use a launch template that specifies an AMI in another AWS account, and the AMI is private and not shared with the AWS account you are using, you receive an error that you're not authorized to use the launch template.

Solution 3: Verify that the permissions on the AMI include the account that you are using. For more information, see Share an AMI with specific AWS accounts in the *Amazon EC2 User Guide*.

Related information

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

Resource	Description
Amazon EC2 Auto Scaling API Reference	The documentation for each API operation shows the request parameters and the XML response and provides links to language-specific SDK reference topics.
autoscaling in the *AWS CLI Command Reference*	Descriptions of the AWS CLI commands that you can use to work with Auto Scaling groups.
AWS Tools for PowerShell Cmdlet Reference	The AWS Tools for PowerShell enable you to script operations on your AWS resources from the PowerShell command line.
Create Auto Scaling groups with AWS CloudFormation	The AWS::AutoScaling::AutoScalingGroup resource lets you build, model, and manage your Auto Scaling groups without manual actions.
Amazon EC2 Auto Scaling endpoints and quotas in the *AWS General Reference*	Information about Amazon EC2 Auto Scaling regions and endpoints.
Product Page	The primary web page for information about Amazon EC2 Auto Scaling.
AWS re:Post	AWS managed question and answer (Q & A) service offering crowd-sourced, expert-re viewed answers to your technical questions.
Create an AMI in the *Amazon EC2 User Guide*	Learn how to create an Amazon Machine Image (AMI) from a customized instance.
How to connect to your Linux instance in the *Amazon EC2 User Guide*	Learn how to connect to the Linux instances that you launch.

Resource	Description
How to connect to your Windows instance in the *Amazon EC2 User Guide*	Learn how to connect to the Windows instances that you launch.
Creating a billing alarm to monitor your estimated AWS charges in the *Amazon CloudWatch User Guide*	Learn how to monitor your estimated charges using CloudWatch.
Application Auto Scaling User Guide	Learn how to configure auto scaling for scalable resources for Amazon Web Services beyond Amazon EC2.

The following general resources are available to help you learn more about AWS.

- Classes & Workshops – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- AWS Developer Center – Explore tutorials, download tools, and learn about AWS developer events.
- AWS Developer Tools – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- Getting Started Resource Center – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- Hands-On Tutorials – Follow step-by-step tutorials to launch your first application on AWS.
- AWS Whitepapers – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- AWS Support Center – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- AWS Support – The primary webpage for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- Contact Us – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.

- AWS Site Terms – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Document history

The following table describes important additions to the Amazon EC2 Auto Scaling documentation, beginning in July 2018. For notification about updates to this documentation, you can subscribe to the RSS feed.

Change	Description	Date
Security IAM update	The AutoScalingServiceRolePolicy managed policy now grants additional permissions to Amazon EC2 (`ec2:GetSecurityGroupsForVpc` and `ec2:GetInstanceTypesFromInstanceRequirements`).	February 29, 2024
Warm pool hibernation supported in additional AWS Regions	You can now hibernate instances in a warm pool in two additional Regions: AWS GovCloud (US-East) and AWS GovCloud (US-West). For more information about warm pools, see Warm pools for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	February 26, 2024
Warm pool hibernation supported in additional AWS Regions	You can now hibernate instances in a warm pool in two additional Regions: Europe (Zurich) and Middle East (UAE). For more information about warm pools, see Warm pools for Amazon EC2 Auto Scaling in	February 21, 2024

	the *Amazon EC2 Auto Scaling User Guide*.	
Support for cross-account parameter use	You can now use an AWS Systems Manager parameter shared from another AWS account with Amazon EC2 Auto Scaling. For more information, see Use AWS Systems Manager parameters instead of AMI IDs in launch templates in the *Amazon EC2 Auto Scaling User Guide*.	February 21, 2024
New Spot price protection option	You can now define your price protection threshold for Spot Instances as a percentage of an On-Demand price when you use attribute-based instance type selection. For more information, see Price protection in the *Amazon EC2 Auto Scaling User Guide*.	January 29, 2024
Instance maintenance policies	You can now use an instance maintenance policy to define whether instances are launched before or after existing instances are terminated during events that cause your instances to be replaced, including an instance refresh. For more information, see Instance maintenance policies in the *Amazon EC2 Auto Scaling User Guide*.	November 15, 2023

Capacity Blocks for ML	You can now launch instances into a Capacity Block by specifying the Capacity Block reservation ID when you create a launch template. With Capacity Blocks, you can reserve GPU instances on a future date to support your short-duration, machine learning (ML) workloads. For more information, see Use Capacity Blocks for machine learning workloads in the *Amazon EC2 Auto Scaling User Guide*.	October 31, 2023
New instance refresh features	You can now configure an instance refresh to set its status to failed and optionally roll back when it detects that a specified CloudWatch alarm has gone into the `ALARM` state. For more informati on, see Undo changes with a rollback in the *Amazon EC2 Auto Scaling User Guide*.	July 31, 2023

Guide changes	A new topic about launching On-Demand Instances into Capacity Reservations has been added to the guide. For more information, see Use On-Demand Capacity Reservations to reserve capacity in specific Availability Zones in the *Amazon EC2 Auto Scaling User Guide*.	July 28, 2023
Guide changes	A new topic about migrating your AWS CloudFormation stacks from launch configurations to launch templates has been added to the guide. For more information, see Migrate AWS CloudFormation stacks from launch configurations to launch templates in the *Amazon EC2 Auto Scaling User Guide*.	April 18, 2023

Support for new API operations	This release adds three new API operations, `AttachTrafficSources`, `DetachTrafficSources`, and `DescribeTrafficSources`. Also, a new field, `TrafficSources`, has been added to the results of `DescribeAutoScalingGroups` operations. A new activity status, `WaitingForConnectionDraining`, has been added to the results of `DescribeScalingActivities` operations. Amazon EC2 Auto Scaling also supports a new value, `VPC_LATTICE`, for the `HealthCheckType` field in `CreateAutoScalingGroup`, `UpdateAutoScalingGroup`, and `DescribeAutoScalingGroups` operations. For more information, see the Amazon EC2 Auto Scaling API Reference.	March 31, 2023
Support for Amazon VPC Lattice	This is the general availability release of VPC Lattice for Amazon EC2 Auto Scaling. For more information, see Route traffic to your Auto Scaling group with a VPC Lattice target group in the *Amazon EC2 Auto Scaling User Guide*.	March 31, 2023

Guide changes	The section with AWS CLI examples for working with Elastic Load Balancing now includes new and updated examples. For more informati on, see Examples for working with Elastic Load Balancing with the AWS Command Line Interface (AWS CLI) in the *Amazon EC2 Auto Scaling User Guide*.	March 31, 2023
Support for predictive scaling in additional AWS Regions	You can now create predictive scaling policies in the Middle East (UAE) and AWS GovCloud (US-East) Regions. For more information, see Predictive scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	March 16, 2023

New instance refresh features	You can now choose to terminate or ignore instances on standby and replace or ignore instances protected from scale in, instead of waiting for them to become replaceable. You can also roll back changes from a failed instance refresh. As part of this update, the documentation has been expanded to include topics for rolling back an instance refresh, cancelling an instance refresh, and understanding the default values for the configurable parameters of an instance refresh. For more information, see Replacing Auto Scaling instances based on an instance refresh in the *Amazon EC2 Auto Scaling User Guide*.	February 10, 2023
Support for using an AWS Systems Manager parameter for an AMI ID	You can now use a Systems Manager parameter instead of an AMI ID in your launch template. For more information, see Using AWS Systems Manager parameters instead of AMI IDs in launch templates in the *Amazon EC2 Auto Scaling User Guide*.	January 19, 2023

Predictive scaling recommendations	You can now get recommendations for evaluating and choosing the right predictive scaling policy from the Amazon EC2 Auto Scaling console. For more information, see Evaluate your predictive scaling policies in the *Amazon EC2 Auto Scaling User Guide*.	January 18, 2023
Predictive scaling forecasts	The forecasts generated by predictive scaling now update every six hours instead of daily. For more information, see Predictive scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	January 6, 2023
Support for CloudWatch metric math	You can now use metric math when you create target tracking scaling policies. With metric math, you can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. For more information, see Create a target tracking scaling policy for Amazon EC2 Auto Scaling using metric math in the *Amazon EC2 Auto Scaling User Guide*.	December 8, 2022

Update to IAM service-linked role permissions	The `AutoScalingServiceRolePolicy` policy now grants additional permissions to Amazon EC2 Auto Scaling. For more information, see AWS managed policies for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	December 6, 2022
New Spot allocation strategy	You can now use the price and capacity optimized allocation strategy to request Spot Instances from the Spot pools that are the least likely to be interrupted and have the lowest possible price. For more information, see Allocation strategies in the *Amazon EC2 Auto Scaling User Guide*.	November 10, 2022
Support for predictive scaling in Asia Pacific (Jakarta) Region	You can now create predictive scaling policies in Asia Pacific (Jakarta) Region. For more information, see Predictive scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	October 13, 2022

Support for custom metrics for predictive scaling in the console	You can now use custom metrics when creating predictive scaling policies from the Amazon EC2 Auto Scaling console. For more information, see Predictive scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	October 13, 2022
CloudWatch monitoring for predictive scaling metrics	You can now access monitorin g data for predictive scaling using CloudWatch. This lets you use metric math to create new time series that display the accuracy of forecast data. For more information, see Monitor predictive scaling metrics with CloudWatch in the *Amazon EC2 Auto Scaling User Guide*.	July 7, 2022
Support for predictive scaling in Asia Pacific (Osaka) Region	You can now create predictive scaling policies in Asia Pacific (Osaka) Region. For more information, see Predictive scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	July 6, 2022

Warm pool hibernation supported in additional Regions	You can now hibernate instances in a warm pool in four additional Regions: Africa (Cape Town), Asia Pacific (Jakarta), Asia Pacific (Osaka), and Europe (Milan). For more information about warm pools, see Warm pools for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	July 5, 2022
Update to health checks	When performing health checks, Amazon EC2 Auto Scaling now helps you minimize any downtime that can occur because of temporary issues or misconfig ured health checks. For more information, see How Amazon EC2 Auto Scaling minimizes downtime in the *Amazon EC2 Auto Scaling User Guide*.	May 21, 2022

Default instance warmup	You can now unify all the warmup and cooldown settings for an Auto Scaling group and optimize the performance of scaling policies that scale continuously by enabling default instance warmup. For more information, see Set the default instance warmup for an Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.	April 19, 2022
Guide changes	A new chapter about integrating with other AWS services has been added to the guide. For more information, see AWS services integrated with Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	March 29, 2022
Update to IAM service-linked role permissions	The `AutoScalingServiceRolePolicy` policy now grants additional read permissions to Amazon EC2 Auto Scaling. For more information, see AWS managed policies for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	March 28, 2022

Instance metadata provides target lifecycle state	You can retrieve an Auto Scaling instance's target lifecycle state from the instance metadata. For more information, see Retrieve the target lifecycle state through instance metadata in the *Amazon EC2 Auto Scaling User Guide*.	March 24, 2022
Support for new warm pool functionality	You can now hibernate instances in a warm pool to stop instances without deleting their memory contents (RAM). You can now also return instances to the warm pool on scale in, instead of always terminati ng instance capacity that you will need later. For more information, see Warm pools for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	February 24, 2022
Guide changes	The Amazon EC2 Auto Scaling console has been updated with additional options to help you start an instance refresh with skip matching enabled and a desired configuration specified. For more information, see Start or cancel an instance refresh (console) in the *Amazon EC2 Auto Scaling User Guide*.	February 3, 2022

Custom metrics for predictive scaling policies	You can now choose whether to use custom metrics when you create predictive scaling policies. You can also use metric math to further customize the metrics that you include in your policy. For more information, see Advanced predictive scaling policy configurations using custom metrics.	November 24, 2021
New On-Demand allocation strategy	You can now choose whether to launch On-Demand Instances based on price (lowest priced instance types first) when you create an Auto Scaling group that uses a mixed instances policy. For more information, see Allocation strategies in the *Amazon EC2 Auto Scaling User Guide*.	October 27, 2021

Attribute-based instance type selection	Amazon EC2 Auto Scaling adds support for attribute-based instance type selection. Instead of manually choosing instance types, you can express your instance requirements as a set of attributes, such as vCPU, memory, and storage. For more information, see Creating an Auto Scaling group using attribute-based instance type selection in the *Amazon EC2 Auto Scaling User Guide*.	October 27, 2021
Support for filtering groups by tags	You can now filter your Auto Scaling groups using tag filters when you retrieve information about your Auto Scaling groups using the `describe-auto-scaling-groups` command. For more information, see Use tags to filter Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.	October 14, 2021

Guide changes	The Amazon EC2 Auto Scaling console has been updated to help you create custom termination policies with AWS Lambda. The console documentation has been revised accordingly. For more information, see Using different termination policies (console).	October 14, 2021
Support for copying launch configurations to launch templates	You can now copy all launch configurations in an AWS Region to new launch templates from the Amazon EC2 Auto Scaling console. For more information, see Copy launch configurations to launch templates in the *Amazon EC2 Auto Scaling User Guide*.	August 9, 2021

Expands instance refresh functionality	You can now include updates, such as a new version of a launch template, when replacing instances by adding your desired configuration to the `start-instance-refresh` command. You can also skip replacing instances that already have your desired configuration by enabling skip matching. For more information, see Replacing Auto Scaling instances based on an instance refresh in the *Amazon EC2 Auto Scaling User Guide*.	August 5, 2021
Support for custom terminati on policies	You can now create custom termination policies with AWS Lambda. For more information, see Creating a custom termination policy with Lambda. The documenta tion for specifying terminati on policies has been updated accordingly.	July 29, 2021
Guide changes	The Amazon EC2 Auto Scaling console has been updated and enhanced with additiona l features to help you create scheduled actions with a time zone specified. The documentation for Scheduled scaling has been revised accordingly.	June 3, 2021

gp3 volumes in launch configurations	You can now specify gp3 volumes in the block device mappings for launch configurations.	June 2, 2021
Support for predictive scaling	You can now use predictive scaling to proactively scale your Amazon EC2 Auto Scaling groups using a scaling policy. For more informati on, see Predictive scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*. With this update, the AutoScali ngServiceRolePolicy managed policy now includes permissio n to call the `cloudwatc h:GetMetricData` API action.	May 19, 2021
Guide changes	You can now access example templates for lifecycle hooks from GitHub. For more information, see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.	April 9, 2021

Support for warm pools	You can now balance performance (minimize cold starts) and cost (stop over-provisioning instance capacity) for applications with long first boot times by adding warm pools to Auto Scaling groups. For more information, see Warm pools for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	April 8, 2021
Support for checkpoints	You can now add checkpoin ts to an instance refresh to replace instances in phases and perform verifications on your instances at specific points. For more informati on, see Adding checkpoints to an instance refresh in the *Amazon EC2 Auto Scaling User Guide*.	March 18, 2021
Guide changes	Improved documentation for using EventBridge with Amazon EC2 Auto Scaling events and lifecycle hooks. For more information, see Using Amazon EC2 Auto Scaling with EventBridge and Tutorial: Configure a lifecycle hook that invokes a Lambda function in the *Amazon EC2 Auto Scaling User Guide*.	March 18, 2021

Support for local time zones	You can now create recurring scheduled actions in the local time zone by adding the `--time-zone` option to the **put-scheduled-update-group-action** command. If your time zone observes Daylight Saving Time (DST), the recurring action automatically adjusts for Daylight Saving Time. For more information, see Scheduled scaling in the *Amazon EC2 Auto Scaling User Guide*.	March 9, 2021
Expands functionality for mixed instances policies	You can now prioritize instance types for your Spot capacity when you use a mixed instances policy. Amazon EC2 Auto Scaling attempts to fulfill prioritie s on a best-effort basis but optimizes for capacity first. For more information, see Auto Scaling groups with multiple instance types and purchase options in the *Amazon EC2 Auto Scaling User Guide*.	March 8, 2021

Scaling activities for deleted groups	You can now view scaling activities for deleted Auto Scaling groups by adding the `--include-deleted-groups` option to the **describe-scaling-activities** command. For more information, see Troubleshooting Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	February 23, 2021
Console improvements	You can now create and attach an Application Load Balancer or Network Load Balancer from the Amazon EC2 Auto Scaling console. For more information, see Create and attach a new Application Load Balancer or Network Load Balancer (console) in the *Amazon EC2 Auto Scaling User Guide*.	November 24, 2020
Multiple network interfaces	You can now configure a launch template for an Auto Scaling group that specifies multiple network interfaces. For more information, see Network interfaces in a VPC.	November 23, 2020

Multiple launch templates	Multiple launch templates can now be used with Auto Scaling groups. For more information, see Specifying a different launch template for an instance type in the *Amazon EC2 Auto Scaling User Guide*.	November 19, 2020
Gateway Load Balancers	Updated guide to show how to attach a Gateway Load Balancer to an Auto Scaling group to ensure that appliance instances launched by Amazon EC2 Auto Scaling are automatic ally registered and deregiste red from the load balancer. For more information, see Elastic Load Balancing types and Attaching a load balancer to your Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.	November 10, 2020
Maximum instance lifetime	You can now reduce the maximum instance lifetime to one day (86,400 seconds). For more information, see Replacing Auto Scaling instances based on maximum instance lifetime in the *Amazon EC2 Auto Scaling User Guide*.	November 9, 2020

Capacity Rebalancing	You can configure your Auto Scaling group to launch a replacement Spot Instance when Amazon EC2 emits a rebalance recommendation. For more information, see Amazon EC2 Auto Scaling Capacity Rebalancing in the *Amazon EC2 Auto Scaling User Guide*.	November 4, 2020
Instance metadata service version 2	You can require the use of Instance Metadata Service Version 2, which is a session-oriented method for requesting instance metadata, when using launch configurations. For more information, see Configuring the instance metadata options in the *Amazon EC2 Auto Scaling User Guide*.	July 28, 2020
Guide changes	Various improvements and new console procedures in the Controlling which Auto Scaling instances terminate during scale in, Monitoring your Auto Scaling instances and groups, Launch templates, and Launch configurations sections of the *Amazon EC2 Auto Scaling User Guide*.	July 28, 2020

Instance refresh	Start an instance refresh to update all instances in your Auto Scaling group when you make a configuration change. For more informati on, see Replacing Auto Scaling instances based on an instance refresh in the *Amazon EC2 Auto Scaling User Guide*.	June 16, 2020
Guide changes	Various improvements in the Replacing Auto Scaling instances based on maximum instance lifetime, Auto Scaling groups with multiple instance types and purchase options, Scaling based on Amazon SQS, and Tagging Auto Scaling groups and instances sections of the *Amazon EC2 Auto Scaling User Guide*.	May 6, 2020
Guide changes	Various improvements to IAM documentation. For more information, see Launch template support and Amazon EC2 Auto Scaling identity-based policy examples in the *Amazon EC2 Auto Scaling User Guide*.	March 4, 2020

Disable scaling policies	You can now disable and re-enable scaling policies. This feature allows you to temporarily disable a scaling policy while preserving the configuration details so that you can enable the policy again later. For more information, see Disabling a scaling policy for an Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.	February 18, 2020
Add notification functionality	Amazon EC2 Auto Scaling now sends events to your AWS Health Dashboard when your Auto Scaling groups cannot scale out due to a missing security group or launch template. For more information, see AWS Health Dashboard notifications for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	February 12, 2020

Guide changes	Various improvements and corrections in the How Amazon EC2 Auto Scaling works with IAM, Amazon EC2 Auto Scaling identity-based policy examples, Required CMK key policy for use with encrypted volumes, and Monitoring your Auto Scaling instances and groups sections of the *Amazon EC2 Auto Scaling User Guide*.	February 10, 2020
Guide changes	Improved documentation for Auto Scaling groups that use instance weighting. Learn how to use scaling policies when using "capacity units" to measure desired capacity. For more information, see How scaling policies work and Scaling adjustment types in the *Amazon EC2 Auto Scaling User Guide*.	February 6, 2020

New "Security" chapter	A new Security chapter in the *Amazon EC2 Auto Scaling User Guide* helps you understand how to apply the shared responsibility model when using Amazon EC2 Auto Scaling. As part of this update, the user guide chapter "Controlling Access to Your Amazon EC2 Auto Scaling Resources" has been replaced by a new, more useful section, Identity and access management for Amazon EC2 Auto Scaling.	February 4, 2020
Recommendations for instance types	AWS Compute Optimizer provides Amazon EC2 instance recommendations to help you improve performan ce, save money, or both. For more information, see Getting recommendations for an instance type in the *Amazon EC2 Auto Scaling User Guide*.	December 3, 2019

Dedicated Hosts and host resource groups	Updated guide to show how to create a launch template that specifies a host resource group. This allows you to create an Auto Scaling group with a launch template that specifies a BYOL AMI to use on Dedicated Hosts. For more information, see Creating a launch template for an Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.	December 3, 2019
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and Amazon EC2 Auto Scaling. For more information, see Amazon EC2 Auto Scaling and interface VPC endpoints in the *Amazon EC2 Auto Scaling User Guide*.	November 22, 2019
Maximum instance lifetime	You can now replace instances automatically by specifyin g the maximum length of time that an instance can be in service. If any instances are approaching this limit, Amazon EC2 Auto Scaling gradually replaces them. For more information, see Replacing Auto Scaling instances based on maximum instance lifetime in the *Amazon EC2 Auto Scaling User Guide*.	November 19, 2019

Instance weighting	For Auto Scaling groups with multiple instance types, you can now optionally specify the number of capacity units that each instance type contributes to the capacity of the group. For more informati on, see Instance weighting for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.	November 19, 2019
Minimum number of instance types	You no longer have to specify additional instance types for groups of Spot, On-Demand , and Reserved Instances. For all Auto Scaling groups, the minimum is now one instance type. For more informati on, see Auto Scaling groups with multiple instance types and purchase options in the *Amazon EC2 Auto Scaling User Guide*.	September 16, 2019

Support for new Spot allocation strategy	Amazon EC2 Auto Scaling now supports a new Spot allocation strategy "capacity -optimized" that fulfills your request using Spot Instance pools that are optimally chosen based on the available Spot capacity. For more information, see Auto Scaling groups with multiple instance types and purchase options in the *Amazon EC2 Auto Scaling User Guide*.	August 12, 2019
Guide changes	Improved Amazon EC2 Auto Scaling documentation in the Service-linked roles and Required CMK key policy for use with encrypted volumes topics.	August 1, 2019
Support for tagging enhancement	Amazon EC2 Auto Scaling now adds tags to Amazon EC2 instances as part of the same API call that launches the instances. For more informati on, see Tagging Auto Scaling groups and instances.	July 26, 2019

Guide changes	Improved Amazon EC2 Auto Scaling documentation in the Suspending and resuming scaling processes topic. Updated Customer managed policy examples to include an example policy that allows users to pass only specific custom suffix service-linked roles to Amazon EC2 Auto Scaling.	June 13, 2019
Support for new Amazon EBS feature	Added support for new Amazon EBS feature in the launch template topic. Change the encryption state of an EBS volume while restoring from a snapshot. For more information, see Creating a launch template for an Auto Scaling group in the *Amazon EC2 Auto Scaling User Guide*.	May 13, 2019
Guide changes	Improved Amazon EC2 Auto Scaling documentation in the following sections: Controlling which Auto Scaling instances terminate during scale in, Auto Scaling groups, Auto Scaling groups with multiple instance types and purchase options, and Dynamic scaling for Amazon EC2 Auto Scaling.	March 12, 2019

Support for combining instance types and purchase options	Provision and automatically scale instances across purchase options (Spot, On-Demand, and Reserved Instances) and instance types within a single Auto Scaling group. For more information, see Auto Scaling groups with multiple instance types and purchase options in the *Amazon EC2 Auto Scaling User Guide*.	November 13, 2018
Updated topic for scaling based on Amazon SQS	Updated guide to explain how you can use custom metrics to scale an Auto Scaling group in response to changing demand from an Amazon SQS queue. For more information, see Scaling based on Amazon SQS in the *Amazon EC2 Auto Scaling User Guide*.	July 26, 2018

The following table describes important changes to the Amazon EC2 Auto Scaling documentation before July 2018.

Feature	Description	Release date
Support for target tracking scaling policies	Set up dynamic scaling for your application in just a few steps. For more information, see Target tracking scaling policies for Amazon EC2 Auto Scaling.	12 July 2017
Support for resource-level permissions	Create IAM policies to control access at the resource level. For more information, see Controlling access to your Amazon EC2 Auto Scaling resources.	15 May 2017

Feature	Description	Release date
Monitoring improvements	Auto Scaling group metrics no longer require that you enable detailed monitoring. You can now enable group metrics collection and view metrics graphs from the **Monitoring** tab in the console. For more information, see Monitoring your Auto Scaling groups and instances using Amazon CloudWatch.	18 August 2016
Support for Application Load Balancers	Attach one or more target groups to a new or existing Auto Scaling group. For more information, see Attaching a load balancer to your Auto Scaling group.	11 August 2016
Events for lifecycle hooks	Amazon EC2 Auto Scaling sends events to EventBrid ge when it calls lifecycle hooks. For more information, see Getting EventBridge when your Auto Scaling group scales.	24 February 2016
Instance protection	Prevent Amazon EC2 Auto Scaling from selecting specific instances for termination when scaling in. For more information, see Instance protection.	07 December 2015
Step scaling policies	Create a scaling policy that enables you to scale based on the size of the alarm breach. For more information, see Scaling policy types.	06 July 2015
Update load balancer	Attach a load balancer to or detach a load balancer from an existing Auto Scaling group. For more informati on, see Attaching a load balancer to your Auto Scaling group.	11 June 2015
Support for ClassicLink	Link EC2-Classic instances in your Auto Scaling group to a VPC, enabling communication between these linked EC2-Classic instances and instances in the VPC using private IP addresses. For more information, see Linking EC2-Classic instances to a VPC.	19 January 2015

Feature	Description	Release date
Lifecycle hooks	Hold your newly launched or terminating instances in a pending state while you perform actions on them. For more information, see Amazon EC2 Auto Scaling lifecycle hooks.	30 July 2014
Detach instances	Detach instances from an Auto Scaling group. For more information, see Detach EC2 instances from your Auto Scaling group.	30 July 2014
Put instances into a Standby state	Put instances that are in an `InService` state into a `Standby` state. For more information, see Temporarily removing instances from your Auto Scaling group.	30 July 2014
Manage tags	Manage your Auto Scaling groups using the AWS Management Console. For more information, see Tagging Auto Scaling groups and instances.	01 May 2014
Support for Dedicated Instances	Launch Dedicated Instances by specifying a placement tenancy attribute when you create a launch configura tion. For more information, see Instance placement tenancy.	23 April 2014
Create a group or launch configuration from an EC2 instance	Create an Auto Scaling group or a launch configuration using an EC2 instance. For information about creating a launch configuration using an EC2 instance, see Creating a launch configuration using an EC2 instance. For information about creating an Auto Scaling group using an EC2 instance, see Creating an Auto Scaling group using an EC2 instance.	02 January 2014
Attach instances	Enable automatic scaling for an EC2 instance by attaching the instance to an existing Auto Scaling group. For more information, see Attach EC2 instances to your Auto Scaling group.	02 January 2014

Feature	Description	Release date
View account limits	View the limits on Auto Scaling resources for your account. For more information, see Auto Scaling limits.	02 January 2014
Console support for Amazon EC2 Auto Scaling	Access Amazon EC2 Auto Scaling using the AWS Management Console. For more information, see Getting started with Amazon EC2 Auto Scaling.	10 December 2013
Assign a public IP address	Assign a public IP address to an instance launched into a VPC. For more information, see Launching Auto Scaling instances in a VPC.	19 September 2013
Instance termination policy	Specify an instance termination policy for Amazon EC2 Auto Scaling to use when terminating EC2 instances. For more information, see Controlling which Auto Scaling instances terminate during scale in.	17 September 2012
Support for IAM roles	Launch EC2 instances with an IAM instance profile. You can use this feature to assign IAM roles to your instances , allowing your applications to access other Amazon Web Services securely. For more information, see Launch Auto Scaling instances with an IAM role.	11 June 2012
Support for Spot Instances	Launch Spot Instances with a launch configuration. For more information, see Requesting Spot Instances for fault-tolerant and flexible applications.	7 June 2012
Tag groups and instances	Tag Auto Scaling groups and specify that the tag also applies to EC2 instances launched after the tag was created. For more information, see Tagging Auto Scaling groups and instances.	26 January 2012

Feature	Description	Release date
Support for Amazon SNS	Use Amazon SNS to receive notifications whenever Amazon EC2 Auto Scaling launches or terminates EC2 instances. For more information, see Getting SNS notifications when your Auto Scaling group scales. Amazon EC2 Auto Scaling also added the following new features: • The ability to set up recurring scaling activities using cron syntax. For more information, see the `PutScheduledUpdateGroupAction` API operation. • A new configuration setting that allows you to scale out without adding the launched instance to the load balancer (LoadBalancer). For more information, see the `ProcessType` API data type. • The `ForceDelete` flag in the `DeleteAutoScalingGroup` operation that tells Amazon EC2 Auto Scaling to delete the Auto Scaling group with the instances associated to it without waiting for the instances to be terminated first. For more information, see the `DeleteAutoScalingGroup` API operation.	20 July 2011
Scheduled scaling actions	Added support for scheduled scaling actions. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling.	2 December 2010
Support for Amazon VPC	Added support for Amazon VPC. For more information, see Launching Auto Scaling instances in a VPC.	2 December 2010
Support for HPC clusters	Added support for high performance computing (HPC) clusters.	2 December 2010

Feature	Description	Release date
Support for health checks	Added support for using Elastic Load Balancing health checks with Amazon EC2 Auto Scaling-managed EC2 instances. For more information, see Health checks for instances in an Auto Scaling group.	2 December 2010
Support for CloudWatch alarms	Removed the older trigger mechanism and redesigned Amazon EC2 Auto Scaling to use the CloudWatch alarm feature. For more information, see Dynamic scaling for Amazon EC2 Auto Scaling.	2 December 2010
Suspend and resume scaling	Added support to suspend and resume scaling processes .	2 December 2010
Support for IAM	Added support for IAM. For more information, see Controlling access to your Amazon EC2 Auto Scaling resources.	2 December 2010