[Auto-scaling](#_Toc13843460)

[Launch Configuration](#_Toc13843461)

[Scheduled Actions](#_Toc13843462)

[Amazon EC2 Auto Scaling Lifecycle Hooks](#_Toc13843463)

[Scaling Policy Types](#_Toc13843464)

[Target tracking scaling](#_Toc13843465)

[Simple and Step Scaling Policies for Amazon EC2 Auto Scaling](#_Toc13843466)

[Health Check Grace Period](#_Toc13843467)

[Termination Policy](#_Toc13843468)

[Suspend-Resume (For troubleshoot)](#_Toc13843469)

[ELB](#_Toc13843470)

[Access logs](#_Toc13843471)

[Steps to Enable Access logs.](#_Toc13843472)

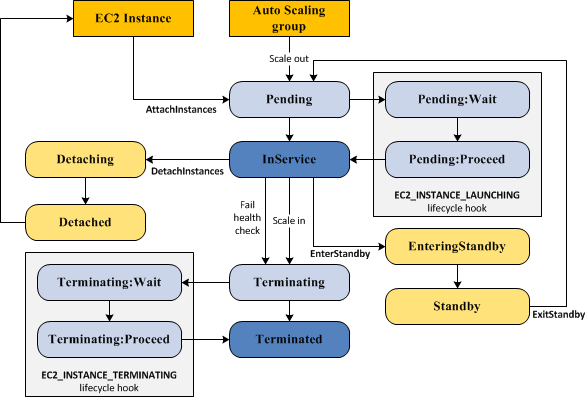
[Health check](#_Toc13843473)

[Scenarios](#_Toc13843474)

[Security](#_Toc13843475)

# Auto-scaling

* Ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application.
* Can detect when an instance is unhealthy, terminate it, and launch an instance to replace it.
* An Auto Scaling group can contain EC2 instances in one or more Availability Zones within the same Region. However, Auto Scaling groups cannot span multiple Regions.



## Launch Configuration

Same the options used to create Ec2 instance but we **will not specify VPC & Subnets**.

## Scheduled Actions

Set your own scaling schedule for predictable load changes.

## Amazon EC2 Auto Scaling Lifecycle Hooks

* Lifecycle hooks enable you to perform custom actions by pausing instances as an Auto Scaling group launches or terminates them.
* When an instance is paused, it remains in a wait state (Pending:Wait or Terminating:Wait) until either you complete the lifecycle action using
  + the complete-lifecycle-action CLI command or
  + CompleteLifecycleAction API action or
  + The timeout period ends (one hour by default).
* Commands
  + --lifecycle-transition autoscaling:EC2\_INSTANCE\_LAUNCHING
  + --lifecycle-transition autoscaling:EC2\_INSTANCE\_TERMINATING
* Each Auto Scaling group can have multiple lifecycle hooks. You can perform a custom action using one or more of the following options:
  + Define a CloudWatch Events target to invoke a Lambda function when a lifecycle action occurs.

Lifecycle Action occured --> CloudWatch Event --> Target.

* Create a notification target and an IAM role. The target can be either an Amazon SQS queue or an Amazon SNS topic. The role allows Amazon EC2 Auto Scaling to publish lifecycle notifications to the target.
* Ec2- User Data

|  |  |
| --- | --- |
| **Name** | **Description** |
| putlifecycle-hook | Creates new Lifecycle hook |
| complete-lifecycleaction | Continue to the next state if you finish before the timeout period ends |
| record-lifecycle-actionheartbeat | Postpone the end of the timeout period |

## Scaling Policy Types

### Target tracking scaling

* Amazon EC2 Auto Scaling creates and manages the CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and the target value.
* 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 cannot use a target tracking scaling policy to scale out your Auto Scaling group when the specified metric is below the target value.
* You can have multiple target tracking scaling policies for an Auto Scaling group, provided that each of them uses a different metric.
  + - EC2 Auto Scaling gives precedence to the largest capacity if both scaling policy starts at same time
* Check the Disable scale-in option to create only a scale-out policy.
* Instance Warmup: The number of seconds that it takes for a newly launched instance to warm up.

### Simple and Step Scaling Policies for Amazon EC2 Auto Scaling

* Step scaling policies increase or decrease the current capacity of your Auto Scaling group based on a set of scaling adjustments, known as step adjustments. The adjustments vary based on the size of the alarm breach.
* In most cases, step scaling policies are a better choice than simple scaling policies.
  + With simple scaling, after a scaling activity is started, the policy must wait for the scaling activity or health check replacement to complete and **the cooldown period** to expire before responding to additional alarms. Cooldown periods help to prevent the initiation of additional scaling activities before the effects of previous activities are visible.

## Health Check Grace Period

* The length of time that Auto Scaling waits before checking an instance's health status. The grace period begins when an instance comes into service.

## Termination Policy

* Old instance
* New Instance
* Old Launch configuration
* Old Launch template
* ClosestToNextInstanceHour
* Allocation Strategy
* Default : **If there are instances in multiple Availability Zones, select the Availability Zone with the most instances.**

## Suspend-Resume (For troubleshoot)

Use the standby feature instead of the suspend-resume feature if you need to troubleshoot or reboot an instance.

* + Launch
  + Terminate
  + AddToLoadBalancer
  + AlarmNotification
  + AZRebalance
  + HealthCheck
  + ReplaceUnhealthy
  + ScheduledActions

# ELB

## Access logs

* Contains

1. Req. Time
2. client Ip address
3. Req path
4. Server response
5. Latencies.
6. No HTTP codes are stored.

* Disabled by default.
* Logs are stored in S3 bucket and auto encrypted/decrypted.
* Does not log health check requests.
* error\_reason field: stores error reason codes.
* Reports metrics to CloudWatch only when requests are flowing through the load balancer and sends its metrics in 60-second intervals.
* UnHealthyHostCount: This metric provides the number of unhealthy instances registered with your load balancer.

## Steps to Enable Access logs.

* Create new s3 bucket.
* Block all public access -> off
* Add new bucket policy , select policy generator...by selecting A/c ID for the region

<https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/enable-access-logs.html>

* Note: /nivikuttoos/AWSLogs/969968687208
* get ELB and , check for access logs and enable it.

## Health check

* Ping
* Attempts Connections
* Send req. to Ec2.

### Scenarios

* ASG only : will check EC2-health status
* ASG + LB but EC2 health check: LB will show instance as outofservice and but ASG shows instance status as healthy.
* ASG + LB but ELB health check: LB will show intance as inservice and flow the traffic.

## Security

* HTTP/HTTPS:
  + Layer 7,
  + can read header from req,
  + x-forwarded-for to get client ip address
  + Only HTTPS can enable sticky session.
* TCP/SSL
  + Layer 4
  + proxy protocol to get client ip address
  + Sticky session can't be enabled.
* SSL Termination: Decrypts and verifies data on the load balancer instead of the application server.
* Connection Draining: Allows existing requests to complete before the load balancer shifts traffic away from a deregistered or unhealthy back-end instance.

# OpsWorks

* Fully managed Chef automate server and You run Chef recipes using Chef Solo.
* **Stack**: Container for AWS resources that have a common purpose and should be logically managed together (Eg: Ec2, RDS etc) . Can also run stack inside VPC.
* **Layers**: A layer represents a set of Amazon EC2 instances that serve a particular purpose, such as serving applications or hosting a database server.
  + For all stacks, AWS OpsWorks Stacks includes service layers, which represent the following AWS services.
    - Amazon Relational Database Service
    - Elastic Load Balancing
    - Amazon Elastic Container Service
* Layers depend on Chef recipes to handle tasks such as installing packages on instances, deploying apps,running scripts, and so on.
* Each layer can have a set of recipes assigned to each lifecycle event (Setup, Configure, Deploy, Undeploy, and Shutdown).
* Instances: An instance represents a single computing resource, such as an Amazon EC2 instance.
  + It defines the resource's basic configuration, such as operating system and size.
  + Other configuration settings, such as Elastic IP addresses or Amazon EBS volumes, are defined by the instance's layers.
  + After the Amazon EC2 instance has finished booting, AWS OpsWorks Stacks installs an agent that handles communication between the instance and the service
  + and runs the appropriate recipes in response to lifecycle events.
  + Instance types :
    - 24/7 instances
    - Time-based instances
    - Load-based instances
  + Autohealing: If an agent stops communicating with the service,AWS OpsWorks Stacks automatically stops and restarts the instance.
* Apps: When you deploy an app, AWS OpsWorks Stacks triggers a **Deploy** event, which runs the Deploy recipes on the stack's instances.
  + You can deploy apps in the following ways:
  + Automatically—When you start instances, AWS OpsWorks Stacks automatically runs the instance's Deploy recipes.
  + Manually—If you have a new app or want to update an existing one, you can manually run the online instances' Deploy recipes.
* OpsWorks Deployment Strategies
  + Version consistency:
    - Issue: Updating the repo will auto update the new instances but not running instances.
    - Solution: Avoid updating master branch and use tagging or Use s3 and enable versioning and rollback.
  + Mannual
    - Run deploy command for apps and 'Update custom cookbooks' for cookbooks all at the same time.
    - Pros: fastest
    - Cons: Downtime
  + Rolling
    - Pros: Less downtime
    - Cons: Failed deployment reduces capacity and req. re deployment.
  + Blue/Green
    - Cons: Cost will high
    - If cost is not a problem, use B/G deployment

# Tools

* AWS Cloud9: is a cloud-based IDE (Pay as you go)
* AWS CodeStar: enables you to quickly develop, build, and deploy applications on AWS. Can operate with Cloud9.(Pay as you go)
* AWS X-Ray : helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. (Pay as you go)
* Git: Distributed version control system (DVCS).

## CodeCommit

* Fully-managed source control service that hosts secure Git-based private repositories.
  + IAM
  + Auto-scalable
  + Encrypted - intransist and at rest.
  + Support PR.
  + Create Repo
* For Authenticating to aws code commit, aws IAM will generate a git username and password. From local machine or CLI, you have to use this credential to connect aws codecommit.

## CodeBuild

* Fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy.
  + Buildspec.yml
  + Auto-scalable
  + processes multiple builds concurrently
  + You are charged by the minute for the compute resources you use.
  + Works with maven, gradle etc.
  + Input can be github/bitbucket/Codecommit/S3
  + Output can be s3 & can communicate with SNS.
  + Logs will send to cloudwatch.

## CodeDeploy:

* Fully managed deployment service.
  + Appspec.yml
* CodeDeploy agent: The CodeDeploy agent is required only if you deploy to an EC2/On-Premises compute platform. The agent is not required for deployments that use the Amazon ECS or AWS Lambda compute platform.
* Auto-scalable
* Can deploy to EC2/Lambda/ECS.
* Revision: Appsepc.yml+ Application (jar) file
* Deployment Types:
  + In-Place deployment: All instance will stopped and install latest revision on each instance and starts.
  + Blue-green deployment

* IAM
* Install CodeDeploy Resource Kit on each Ec2 instances by user data scripts.
* CodeDeploy will deploy application to new instances inside ASG.

CodePipeline:

* Fully managed continuous delivery service (pay as you go).
* Dev commit --> Codepipeline auto detect the changes --> build and run test --> Deployed to staging server --> Run integration and load test ---> Mannual Approval --> Deploy to production servers.
* Can deploy apps to Ec2 instance via
  + CodeDeploy
  + EBS
  + 3)OpsWorks
  + ECS
* Region Based