Contents

[AWS VPC, Regions, Availability Zones & Local Zones 10](#_Toc132921324)

[Virtual Private Cloud (VPC) 10](#_Toc132921325)

[Subnets 10](#_Toc132921326)

[Public and Private Subnet 11](#_Toc132921327)

[Internet Gateway 11](#_Toc132921328)

[Route tables 12](#_Toc132921329)

[NAT Gateway 13](#_Toc132921330)

[Pricing 13](#_Toc132921331)

[NAT Instance 14](#_Toc132921332)

[Regions, Availability Zones and Local Zones 15](#_Toc132921333)

[AWS Regions 16](#_Toc132921334)

[Availability Zones 17](#_Toc132921335)

[Local Zones 17](#_Toc132921336)

[High Availability (HA) with Multi-Region and Multi-AZ 18](#_Toc132921337)

[Create Windows EC2 Instance with Web Server 19](#_Toc132921338)

[Create Custom AMI 20](#_Toc132921339)

[Create an Amazon EBS-backed Linux AMI 20](#_Toc132921340)

[Launch an instance from an AMI you created 23](#_Toc132921341)

[Install LAMP Web Server on Amazon Linux 2023 23](#_Toc132921342)

[Step 1: Prepare the LAMP server 23](#_Toc132921343)

[Step 2: Test your LAMP server 26](#_Toc132921344)

[Step 3: Secure the database server (optional) 27](#_Toc132921345)

[Step 4: (Optional) Install phpMyAdmin 28](#_Toc132921346)

[Create AMI from LAMP server instance 30](#_Toc132921347)

[Amazon Identity and Access Management (IAM) 31](#_Toc132921348)

[IAM Roles 31](#_Toc132921349)

[Common scenarios for roles: Users, applications, and services 31](#_Toc132921350)

[Providing access to an AWS service 32](#_Toc132921351)

[Identities: Roles, Users and Federated Identities 32](#_Toc132921352)

[AWS Users 32](#_Toc132921353)

[Federated Identities 33](#_Toc132921354)

[IAM Roles 33](#_Toc132921355)

[Structure of an IAM role 33](#_Toc132921356)

[Trust policy 34](#_Toc132921357)

[Permission policy 34](#_Toc132921358)

[Creating Our First Role 34](#_Toc132921359)

[Creating our first role 34](#_Toc132921360)

[Assuming IAM Roles 36](#_Toc132921361)

[Using an IAM role to create resources 37](#_Toc132921362)

[To stop using a role (AWS Console) 38](#_Toc132921363)

[Amazon Simple Storage Service (S3) 39](#_Toc132921364)

[Create a Static Web Site 39](#_Toc132921365)

[S3 Bucket Policy 40](#_Toc132921366)

[Getting a List of Objects in a Bucket with a Specific Prefix 40](#_Toc132921367)

[Allow a User to Read Only Objects That Have a Specific Tag Key and Value 42](#_Toc132921368)

[Allow a User to Only Add Objects with a Specific Object Tag Key and Value 42](#_Toc132921369)

[Restrict Access to Specific IP Addresses 43](#_Toc132921370)

[Create VPC with IGW and NGW 45](#_Toc132921371)

[Create VPCs Peering Connection 47](#_Toc132921372)

[Create VPC, Subnets and Instance in US 47](#_Toc132921373)

[Create VPC, Subnets and Instance in Europe (EU) 48](#_Toc132921374)

[Configure VPC Peering 49](#_Toc132921375)

[Amazon VPC Quotas 51](#_Toc132921376)

[VPC and subnets 51](#_Toc132921377)

[DNS 51](#_Toc132921378)

[Elastic IP addresses 51](#_Toc132921379)

[Gateways 52](#_Toc132921380)

[Network ACLs 52](#_Toc132921381)

[Network interfaces 52](#_Toc132921382)

[Route tables 53](#_Toc132921383)

[Security groups 53](#_Toc132921384)

[VPC peering connections 53](#_Toc132921385)

[VPC endpoints 54](#_Toc132921386)

[Network Access Control List (NACL) 55](#_Toc132921387)

[The Stateless Beauty of AWS NACLs 55](#_Toc132921388)

[Control traffic to subnets using Network ACLs 55](#_Toc132921389)

[Network ACL basics 55](#_Toc132921390)

[Network ACL rules 56](#_Toc132921391)

[Default network ACL 56](#_Toc132921392)

[Work with network ACLs 57](#_Toc132921393)

[Determine network ACL associations 57](#_Toc132921394)

[Create a network ACL 57](#_Toc132921395)

[Add and delete rules 58](#_Toc132921396)

[Associate a subnet with a network ACL 59](#_Toc132921397)

[Disassociate a network ACL from a subnet 59](#_Toc132921398)

[Change a subnet's network ACL 59](#_Toc132921399)

[Delete a network ACL 59](#_Toc132921400)

[Create a MySQL RDS 61](#_Toc132921401)

[Create the DB 61](#_Toc132921402)

[Connect to the DB 61](#_Toc132921403)

[VPC with EC2 Instance and RDS 62](#_Toc132921404)

[Create a VPC for use with a DB instance (dual-stack mode) 62](#_Toc132921405)

[Create a VPC with private and public subnets 63](#_Toc132921406)

[Create a VPC security group for a public web server 63](#_Toc132921407)

[Create a VPC security group for a private DB instance 64](#_Toc132921408)

[Create a DB subnet group 65](#_Toc132921409)

[Create a web server and an Amazon RDS DB instance 66](#_Toc132921410)

[Launch an EC2 instance 66](#_Toc132921411)

[Create a DB instance 67](#_Toc132921412)

[Install a web server on your EC2 instance 68](#_Toc132921413)

[Enabling Automated Backups for RDS 75](#_Toc132921414)

[Viewing automated backups 75](#_Toc132921415)

[Retaining automated backups 75](#_Toc132921416)

[Disabling automated backups 75](#_Toc132921417)

[Enabling cross-Region automated backups 76](#_Toc132921418)

[Stopping automated backup replication 77](#_Toc132921419)

[Deleting replicated backups 77](#_Toc132921420)

[Creating a DB snapshot 78](#_Toc132921421)

[Restoring from a DB snapshot 79](#_Toc132921422)

[Deleting a DB snapshot 80](#_Toc132921423)

[Backing up and restoring a Multi-AZ DB cluster 80](#_Toc132921424)

[Creating a Multi-AZ DB cluster snapshot 80](#_Toc132921425)

[Restoring from a snapshot to a Multi-AZ DB cluster 81](#_Toc132921426)

[Working with MySQL read replicas 81](#_Toc132921427)

[Load Balancer 83](#_Toc132921428)

[Auto Scaling 86](#_Toc132921429)

[Different AWS AutoScaling Policy 86](#_Toc132921430)

[Dynamic Scaling 86](#_Toc132921431)

[Predictive scaling 86](#_Toc132921432)

[Scheduled scaling 86](#_Toc132921433)

[Differences between step scaling policies and simple scaling policies 87](#_Toc132921434)

[Simple Scaling 87](#_Toc132921435)

[Step Scaling 87](#_Toc132921436)

[Summary 88](#_Toc132921437)

[Demo: Configuration of an AutoScaling Group 88](#_Toc132921438)

[Step-1: Choose launch template or configuration 88](#_Toc132921439)

[Step-2: Choose instance launch options 89](#_Toc132921440)

[Step-3: Configure advanced options (optional) 89](#_Toc132921441)

[Step-4: Configure group size and scaling policies 89](#_Toc132921442)

[Step-5: Add notifications (optional) 90](#_Toc132921443)

[Step-6: Add tags (optional) 90](#_Toc132921444)

[Step-7: Review the AWS AutoScaling Policy configuration 90](#_Toc132921445)

[Verify and Monitor AWS AutoScaling Policy Behaviour 90](#_Toc132921446)

[Scaling In / Down 91](#_Toc132921447)

[Elastic Block Store (EBS) and Volumes 93](#_Toc132921448)

[Features of Amazon EBS 93](#_Toc132921449)

[Amazon EBS Volumes 94](#_Toc132921450)

[View your current limits 94](#_Toc132921451)

[Amazon EBS volume types 95](#_Toc132921452)

[Solid state drive (SSD) volumes 95](#_Toc132921453)

[Hard disk drive (HDD) volumes 95](#_Toc132921454)

[Previous generation volumes 95](#_Toc132921455)

[Amazon EBS snapshots 95](#_Toc132921456)

[Amazon EBS vs. Ephemeral Storage 95](#_Toc132921457)

[Demo: Increase the size of an Amazon EBS volume on an EC2 instance 96](#_Toc132921458)

[Step 1: Launch an EC2 instance with an added EBS volume 97](#_Toc132921459)

[Step 2: Make the data volume available for use 97](#_Toc132921460)

[Step 3: Increase the size of the data volume 99](#_Toc132921461)

[Step 4: Extend the file system 100](#_Toc132921462)

[Route 53 101](#_Toc132921463)

[What is Amazon Route53? 101](#_Toc132921464)

[1. Register domain names 101](#_Toc132921465)

[2. Route internet traffic to the resources for your domain 101](#_Toc132921466)

[3. Check the health of your resources 101](#_Toc132921467)

[How Amazon Route 53 routes traffic for your domain 101](#_Toc132921468)

[Simple routing policy 103](#_Toc132921469)

[Failover routing policy 103](#_Toc132921470)

[Geolocation routing policy 103](#_Toc132921471)

[Geoproximity routing policy 103](#_Toc132921472)

[Latency routing policy (LBR) 104](#_Toc132921473)

[IP-based routing policy 105](#_Toc132921474)

[Multivalue answer routing policy 105](#_Toc132921475)

[Weighted routing policy 106](#_Toc132921476)

[How Amazon Route 53 checks the health of your resources 106](#_Toc132921477)

[Configure Health Check for EC2 Instances 107](#_Toc132921478)

[AWS CloudWatch – Monitoring 109](#_Toc132921479)

[Dashboard 110](#_Toc132921480)

[Monitor EBS 110](#_Toc132921481)

[Billing Alarm 110](#_Toc132921482)

[Creating a billing alarm 110](#_Toc132921483)

[Amazon CloudWatch Logs concepts 111](#_Toc132921484)

[Log events 112](#_Toc132921485)

[Log streams 112](#_Toc132921486)

[Log groups 112](#_Toc132921487)

[Metric filters 112](#_Toc132921488)

[Retention settings 115](#_Toc132921489)

[Enable Logging on EC2 Instance 115](#_Toc132921490)

[Serverless Computing 117](#_Toc132921491)

[AWS Lambda 117](#_Toc132921492)

[AWS Fargate 117](#_Toc132921493)

[Amazon SNS 118](#_Toc132921494)

[Amazon SQS 118](#_Toc132921495)

[AWS DynamoDB 118](#_Toc132921496)

[Amazon Aurora 118](#_Toc132921497)

[AWS Step Functions 119](#_Toc132921498)

[Amazon API Gateway 119](#_Toc132921499)

[Use case: 119](#_Toc132921500)

[Step1: Create a DynamoDB table 120](#_Toc132921501)

[Serverless Lambda Functions with NodeJS 124](#_Toc132921502)

[To create an execution role 124](#_Toc132921503)

[To create a Node.js function 124](#_Toc132921504)

[Sample NodeJS Lambda code 124](#_Toc132921505)

[Lambda Function -> REST API 125](#_Toc132921506)

[Using Lambda with API Gateway - NodeJS 126](#_Toc132921507)

[Prerequisites 127](#_Toc132921508)

[Create a permissions policy 129](#_Toc132921509)

[Create an execution role 130](#_Toc132921510)

[Create the function 131](#_Toc132921511)

[Invoke the function using the AWS CLI 133](#_Toc132921512)

[Create a REST API using API Gateway 134](#_Toc132921513)

[Create a resource on your REST API 134](#_Toc132921514)

[Create an HTTP POST method 135](#_Toc132921515)

[Create a DynamoDB table 135](#_Toc132921516)

[Test the integration of API Gateway, Lambda, and DynamoDB 136](#_Toc132921517)

[Deploy the API 138](#_Toc132921518)

[Use curl to invoke your function using HTTP requests 138](#_Toc132921519)

[Clean up your resources (optional) 139](#_Toc132921520)

[Serverless Lambda Functions with Python 141](#_Toc132921521)

[Using Lambda with API Gateway - Python 141](#_Toc132921522)

[Prerequisites 142](#_Toc132921523)

[Create a permissions policy 143](#_Toc132921524)

[Create an execution role 144](#_Toc132921525)

[Create the function 145](#_Toc132921526)

[Invoke the function using the AWS CLI 147](#_Toc132921527)

[Create a REST API using API Gateway 148](#_Toc132921528)

[Create a resource on your REST API 148](#_Toc132921529)

[Create an HTTP POST method 149](#_Toc132921530)

[Create a DynamoDB table 149](#_Toc132921531)

[Test the integration of API Gateway, Lambda, and DynamoDB 150](#_Toc132921532)

[Deploy the API 152](#_Toc132921533)

[Use curl to invoke your function using HTTP requests 152](#_Toc132921534)

[Clean up your resources (optional) 153](#_Toc132921535)

[AWS CLI and SDK 155](#_Toc132921536)

[AWS CLI Installation 155](#_Toc132921537)

[Set and view configuration settings using commands 156](#_Toc132921538)

[Manually editing the credentials and config files 157](#_Toc132921539)

[Environment Variables 158](#_Toc132921540)

[Create an EC2 Linux Instance 158](#_Toc132921541)

[AWS SDK for JavaScript / NodeJS 159](#_Toc132921542)

[Getting Started in Node.js 159](#_Toc132921543)

[Amazon DynamoDB Examples 162](#_Toc132921544)

[Creating and Using Tables in DynamoDB 162](#_Toc132921545)

[Reading and Writing A Single Item in DynamoDB 166](#_Toc132921546)

[Querying and Scanning a DynamoDB Table 170](#_Toc132921547)

[Amazon EC2 Examples 172](#_Toc132921548)

[About the Example 173](#_Toc132921549)

[AWS SDK for Python 174](#_Toc132921550)

[AWS DevOps – CodeCommit, CodeBuild, CodeDeploy, CodePipeline, CodeStar 175](#_Toc132921551)

[Deployment and Provisioning 176](#_Toc132921552)

[Amazon Elastic Beanstalk 176](#_Toc132921553)

[What is AWS Elastic Beanstalk? 176](#_Toc132921554)

[Elastic Beanstalk concepts 176](#_Toc132921555)

[Web server environments 177](#_Toc132921556)

[Worker environments 179](#_Toc132921557)

[AWS CloudFormation 180](#_Toc132921558)

[Simplify infrastructure management 180](#_Toc132921559)

[Quickly replicate your infrastructure 181](#_Toc132921560)

[Easily control and track changes to your infrastructure 181](#_Toc132921561)

[AWS CloudFormation concepts 181](#_Toc132921562)

[Templates 182](#_Toc132921563)

[Stacks 184](#_Toc132921564)

[Change sets 184](#_Toc132921565)

[Installation of LAMP Server in EC2 through CloudFormation 185](#_Toc132921566)

[Create a VPC with Public and Private Subnets 185](#_Toc132921567)

[Launch an EC2 Instance in a VPC 187](#_Toc132921568)

[Querying for the latest AMI using public parameters 187](#_Toc132921569)

[Walkthrough: Create a scaled and load-balanced application 188](#_Toc132921570)

[Full stack template 188](#_Toc132921571)

[Template walkthrough 192](#_Toc132921572)

[Step 1: Launch the stack 194](#_Toc132921573)

[Step 2: Clean up your sample resources 194](#_Toc132921574)

[AWS Cloud Development Kit (CDK) 195](#_Toc132921575)

[Why use the AWS CDK? 196](#_Toc132921576)

[TypeScript 196](#_Toc132921577)

[Python 197](#_Toc132921578)

[C# 197](#_Toc132921579)

[Docker – ECS, ECR 200](#_Toc132921580)

[A Complete Guide to Deploying a Node.js app on AWS with Docker 200](#_Toc132921581)

[Kubernees – EKS 201](#_Toc132921582)

[Well Architected Framework 202](#_Toc132921583)

[The 6 Pillars of the AWS Well-Architected Framework 203](#_Toc132921584)

[1. Operational Excellence 204](#_Toc132921585)

[2. Security 204](#_Toc132921586)

[3. Reliability 205](#_Toc132921587)

[4. Performance Efficiency 206](#_Toc132921588)

[5. Cost Optimization 206](#_Toc132921589)

[6. Sustainability 207](#_Toc132921590)

[Cloud Best Practices 209](#_Toc132921591)

[Perform operations as code 209](#_Toc132921592)

[Use version control for configuration as well as code 209](#_Toc132921593)

[Architect for security 209](#_Toc132921594)

[Learn from operational failures and share learnings across the organization 210](#_Toc132921595)

[Top 10 Practices for Managing the Cloud 210](#_Toc132921596)

[1. Create a Cloud Center of Excellence (CCoE) for Cloud Application Management 210](#_Toc132921597)

[2. Assess Business Goals and Understand the Benefits 210](#_Toc132921598)

[3. Select the Best Model 211](#_Toc132921599)

[4. Understand the Distinct Areas of Cloud Adoption 211](#_Toc132921600)

[5. Establish Governance for Managing Cloud Services 211](#_Toc132921601)

[6. Continuously Optimize Processes 211](#_Toc132921602)

[7. Ensure Data is Actionable 212](#_Toc132921603)

[8. Prioritize Communication 212](#_Toc132921604)

[9. Take Advantage of Automation 212](#_Toc132921605)

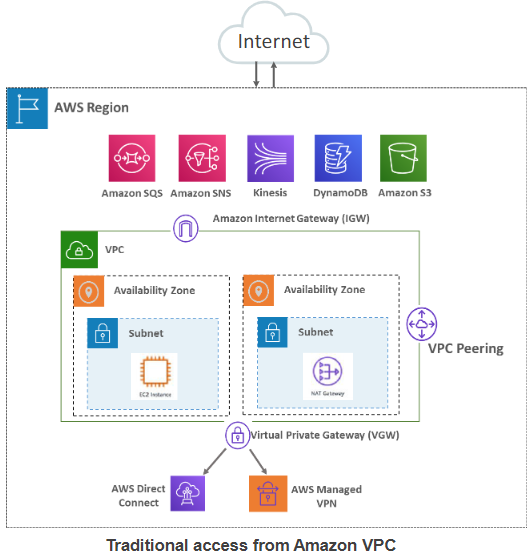
[10. Get Professional Assistance 212](#_Toc132921606)

# AWS VPC, Regions, Availability Zones & Local Zones

## Virtual Private Cloud (VPC)

Amazon Virtual Private Cloud (VPC) is a service that lets you launch AWS resources in a logically isolated virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways. You can use both IPv4 and IPv6 for most resources in your VPC, helping to ensure secure and easy access to resources and applications.

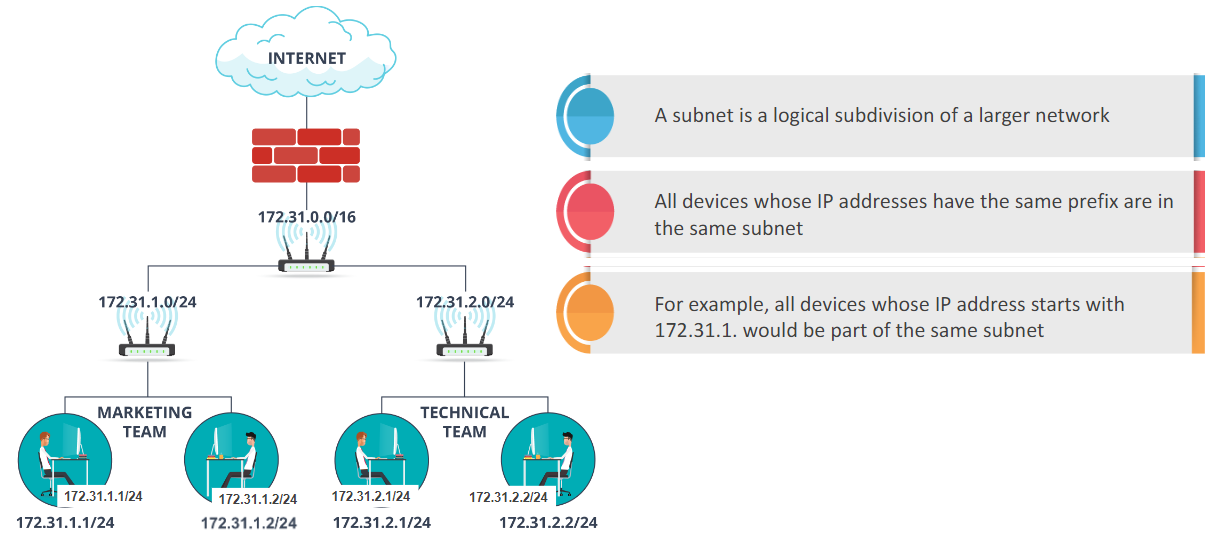
As one of AWS's foundational services, Amazon VPC makes it easy to customize your VPC's network configuration. You can create a public-facing subnet for your web servers that have access to the internet. It also lets you place your backend systems, such as databases or application servers, in a private-facing subnet with no internet access. Amazon VPC lets you to use multiple layers of security, including security groups and network access control lists, to help control access to Amazon Elastic Compute Cloud (Amazon EC2) instances in each subnet.



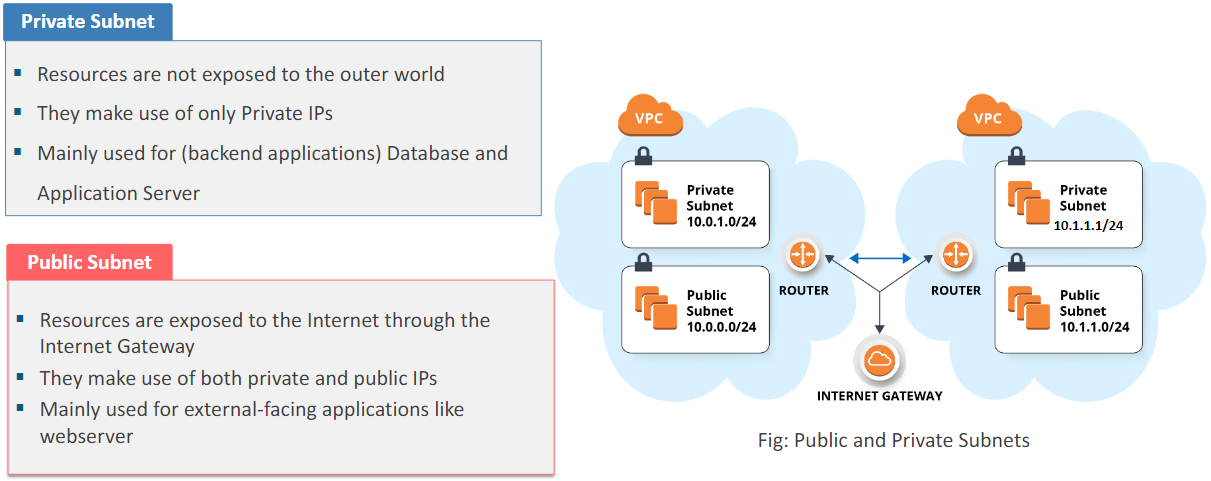
## Subnets

A virtual private cloud (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud. You can specify an IP address range for the VPC, add subnets, add gateways, and associate security groups.

A subnet is a range of IP addresses in your VPC. You launch AWS resources, such as Amazon EC2 instances, into your subnets. You can connect a subnet to the internet, other VPCs, and your own data centers, and route traffic to and from your subnets using route tables.



### Public and Private Subnet



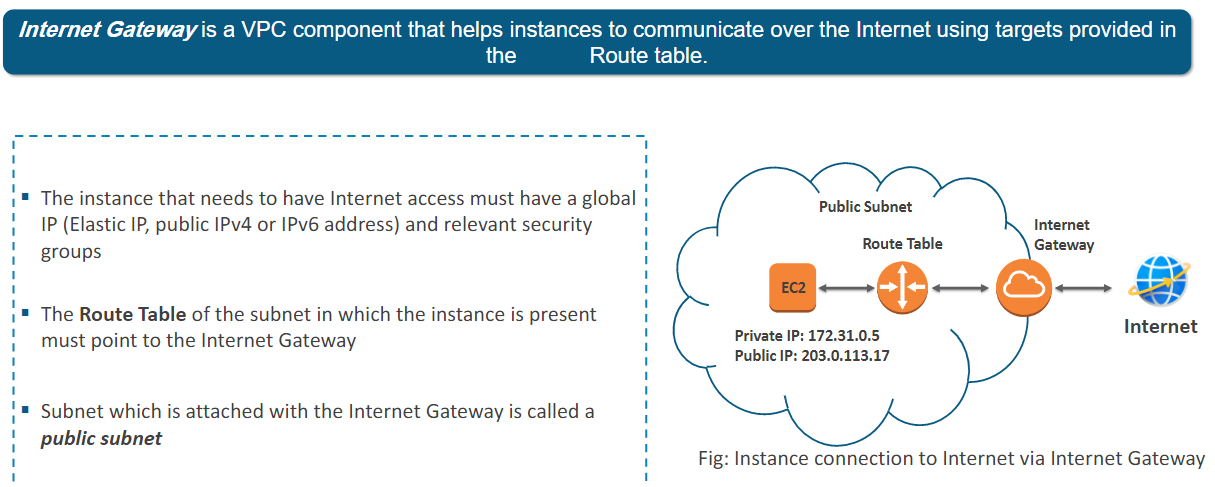
## Internet Gateway

An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between your VPC and the internet. It supports IPv4 and IPv6 traffic. It does not cause availability risks or bandwidth constraints on your network traffic.

An internet gateway enables resources in your public subnets (such as EC2 instances) to connect to the internet if the resource has a public IPv4 address or an IPv6 address. Similarly, resources on the internet can initiate a connection to resources in your subnet using the public IPv4 address or IPv6 address. For example, an internet gateway enables you to connect to an EC2 instance in AWS using your local computer.

An internet gateway provides a target in your VPC route tables for internet-routable traffic. For communication using IPv4, the internet gateway also performs network address translation (NAT). For communication using IPv6, NAT is not needed because IPv6 addresses are public. For more information, see [IP addresses and NAT](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html#ip-addresses-and-nat).

There's no additional charge for creating an internet gateway.




                Using an internet gateway
            

## Route tables

A route table contains a set of rules, called routes, that are used to determine where network traffic from your VPC is directed. You can explicitly associate a subnet with a particular route table. Otherwise, the subnet is implicitly associated with the main route table.

Each route in a route table specifies the range of IP addresses where you want the traffic to go (the destination) and the gateway, network interface, or connection through which to send the traffic (the target).


   A resource map showing a VPC with subnets in two Availability Zones, three route
    tables, an internet gateway, and a gateway endpoint.
  

## NAT Gateway

A NAT gateway is a Network Address Translation (NAT) service. You can use a NAT gateway so that instances in a private subnet can connect to services outside your VPC but external services cannot initiate a connection with those instances.

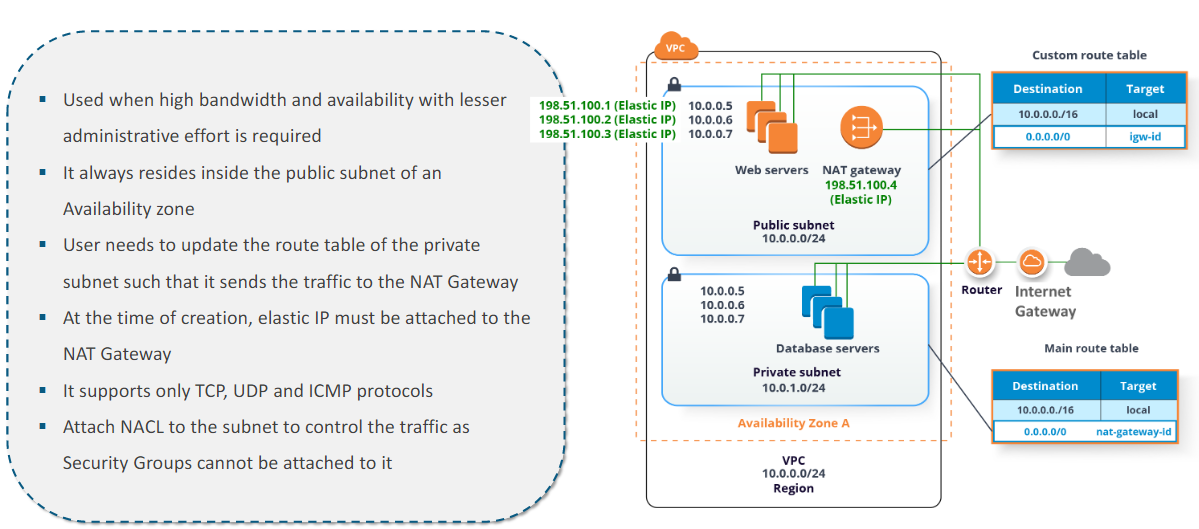
When you create a NAT gateway, you specify one of the following connectivity types:

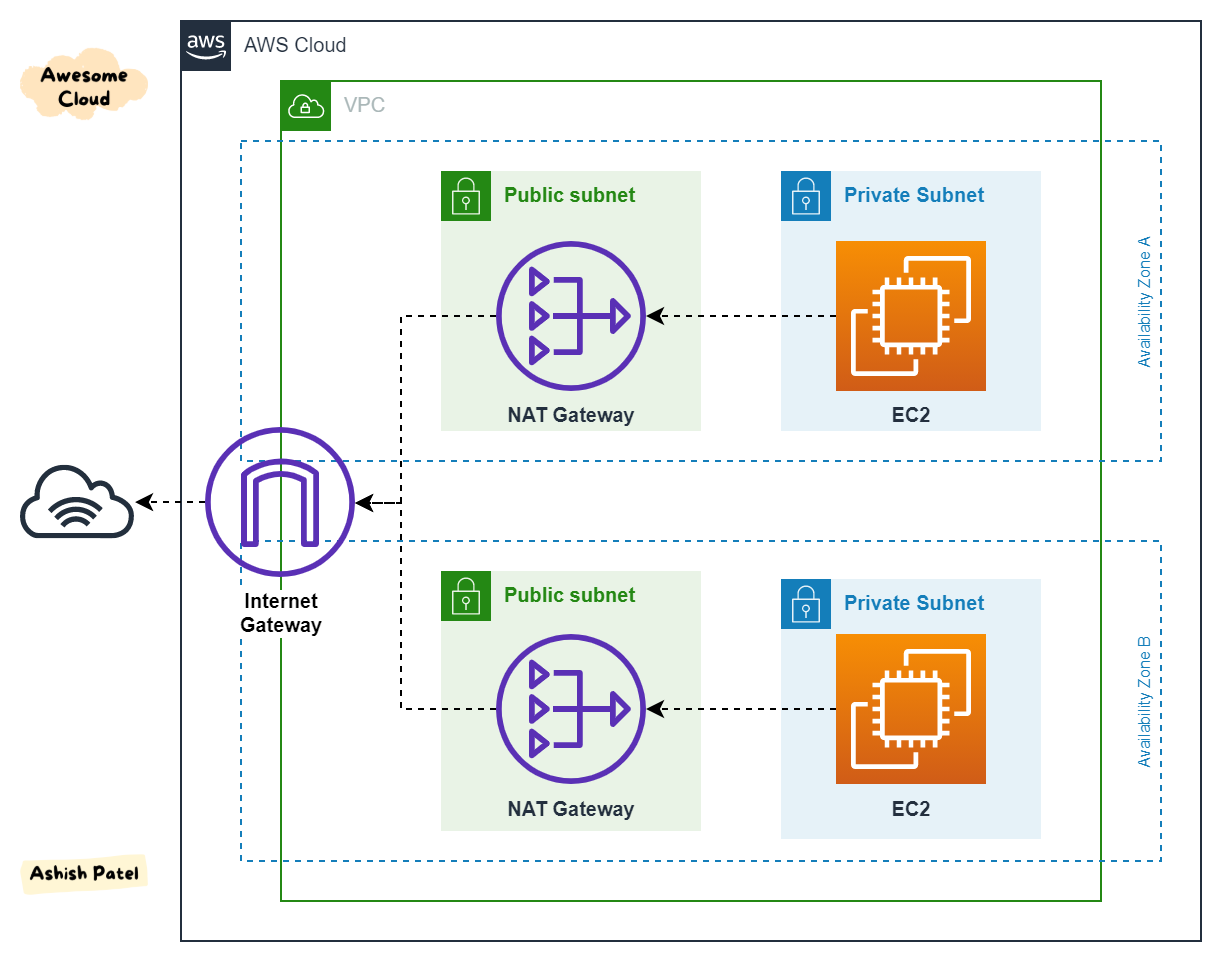
* **Public** – (Default) Instances in private subnets can connect to the internet through a public NAT gateway, but cannot receive unsolicited inbound connections from the internet. You create a public NAT gateway in a public subnet and must associate an elastic IP address with the NAT gateway at creation. You route traffic from the NAT gateway to the internet gateway for the VPC. Alternatively, you can use a public NAT gateway to connect to other VPCs or your on-premises network. In this case, you route traffic from the NAT gateway through a transit gateway or a virtual private gateway.
* **Private** – Instances in private subnets can connect to other VPCs or your on-premises network through a private NAT gateway. You can route traffic from the NAT gateway through a transit gateway or a virtual private gateway. You cannot associate an elastic IP address with a private NAT gateway. You can attach an internet gateway to a VPC with a private NAT gateway, but if you route traffic from the private NAT gateway to the internet gateway, the internet gateway drops the traffic.

The NAT gateway replaces the source IP address of the instances with the IP address of the NAT gateway. For a public NAT gateway, this is the elastic IP address of the NAT gateway. For a private NAT gateway, this is the private IPv4 address of the NAT gateway. When sending response traffic to the instances, the NAT device translates the addresses back to the original source IP address.

### Pricing

When you provision a NAT gateway, you are charged for each hour that your NAT gateway is available and each Gigabyte of data that it processes. For more information, see [Amazon VPC Pricing](http://aws.amazon.com/vpc/pricing/)

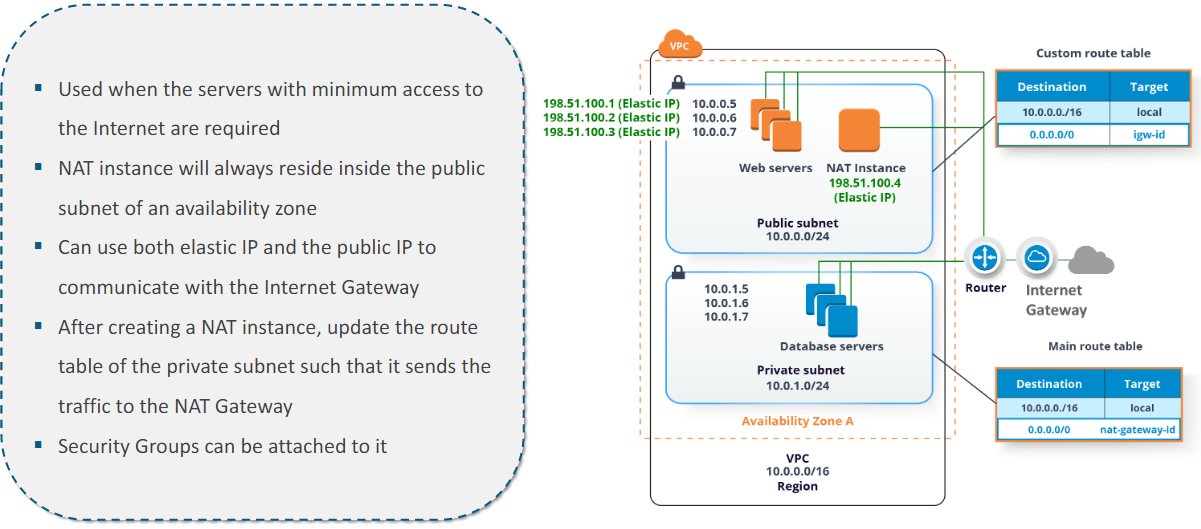




## NAT Instance

You can create your own AMI that provides network address translation and use your AMI to launch an EC2 instance as a NAT instance. You launch a NAT instance in a public subnet to enable instances in the private subnet to initiate outbound IPv4 traffic to the internet or other AWS services, but prevent the instances from receiving inbound traffic initiated on the internet.

Your NAT instance quota depends on your instance quota for the Region. For more information, see [Amazon EC2 service quotas](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-resource-limits.html) in the Amazon EC2 User Guide for Linux Instances.




        NAT instance setup
      

## Regions, Availability Zones and Local Zones

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html>

Amazon cloud computing resources are hosted in multiple locations world-wide. These locations are composed of AWS Regions, Availability Zones, and Local Zones. Each AWS Region is a separate geographic area. Each AWS Region has multiple, isolated locations known as Availability Zones.

By using Local Zones, you can place resources, such as compute and storage, in multiple locations closer to your users. Amazon RDS enables you to place resources, such as DB instances, and data in multiple locations. Resources aren't replicated across AWS Regions unless you do so specifically.

Amazon operates state-of-the-art, highly-available data centers. Although rare, failures can occur that affect the availability of DB instances that are in the same location. If you host all your DB instances in one location that is affected by such a failure, none of your DB instances will be available.


   AWS Region
  

It is important to remember that each AWS Region is completely independent. Any Amazon RDS activity you initiate (for example, creating database instances or listing available database instances) runs only in your current default AWS Region. The default AWS Region can be changed in the console, or by setting the [AWS\_DEFAULT\_REGION](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-region) environment variable. Or it can be overridden by using the --region parameter with the AWS Command Line Interface (AWS CLI). For more information, see [Configuring the AWS Command Line Interface](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html), specifically the sections about environment variables and command line options.

Amazon RDS supports special AWS Regions called AWS GovCloud (US). These are designed to allow US government agencies and customers to move more sensitive workloads into the cloud. The AWS GovCloud (US) Regions address the US government's specific regulatory and compliance requirements. For more information, see [What is AWS GovCloud (US)?](https://docs.aws.amazon.com/govcloud-us/latest/UserGuide/whatis.html)

To create or work with an Amazon RDS DB instance in a specific AWS Region, use the corresponding regional service endpoint.

### AWS Regions

Each AWS Region is designed to be isolated from the other AWS Regions. This design achieves the greatest possible fault tolerance and stability.

When you view your resources, you see only the resources that are tied to the AWS Region that you specified. This is because AWS Regions are isolated from each other, and we don't automatically replicate resources across AWS Regions.

### Availability Zones

When you create a DB instance, you can choose an Availability Zone or have Amazon RDS choose one for you randomly. An Availability Zone is represented by an AWS Region code followed by a letter identifier (for example, us-east-1a).

Use the [describe-availability-zones](https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-availability-zones.html) Amazon EC2 command as follows to describe the Availability Zones within the specified Region that are enabled for your account.

aws ec2 describe-availability-zones --region *region-name*

For example, to describe the Availability Zones within the US East (N. Virginia) Region (us-east-1) that are enabled for your account, run the following command:

aws ec2 describe-availability-zones --region us-east-1

You can't choose the Availability Zones for the primary and secondary DB instances in a Multi-AZ DB deployment. Amazon RDS chooses them for you randomly. For more information about Multi-AZ deployments, see [Configuring and managing a Multi-AZ deployment](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html).

**Note**: Random selection of Availability Zones by RDS doesn't guarantee an even distribution of DB instances among Availability Zones within a single account or DB subnet group. You can request a specific AZ when you create or modify a Single-AZ instance, and you can use more-specific DB subnet groups for Multi-AZ instances. For more information, see [Creating an Amazon RDS DB instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html) and [Modifying an Amazon RDS DB instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.DBInstance.Modifying.html).

### Local Zones

A Local Zone is an extension of an AWS Region that is geographically close to your users. You can extend any VPC from the parent AWS Region into Local Zones. To do so, create a new subnet and assign it to the AWS Local Zone. When you create a subnet in a Local Zone, your VPC is extended to that Local Zone. The subnet in the Local Zone operates the same as other subnets in your VPC.

When you create a DB instance, you can choose a subnet in a Local Zone. Local Zones have their own connections to the internet and support AWS Direct Connect. Thus, resources created in a Local Zone can serve local users with very low-latency communications. For more information, see [AWS Local Zones](http://aws.amazon.com/about-aws/global-infrastructure/localzones/).

A Local Zone is represented by an AWS Region code followed by an identifier that indicates the location, for example us-west-2-lax-1a.

**Note**: A Local Zone can't be included in a Multi-AZ deployment.

To use a Local Zone

1. Enable the Local Zone in the Amazon EC2 console.

For more information, see [Enabling Local Zones](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#enable-zone-group) in the Amazon EC2 User Guide for Linux Instances.

1. Create a subnet in the Local Zone.

For more information, see [Creating a subnet in your VPC](https://docs.aws.amazon.com/vpc/latest/userguide/working-with-vpcs.html#AddaSubnet) in the Amazon VPC User Guide.

1. Create a DB subnet group in the Local Zone.

When you create a DB subnet group, choose the Availability Zone group for the Local Zone.

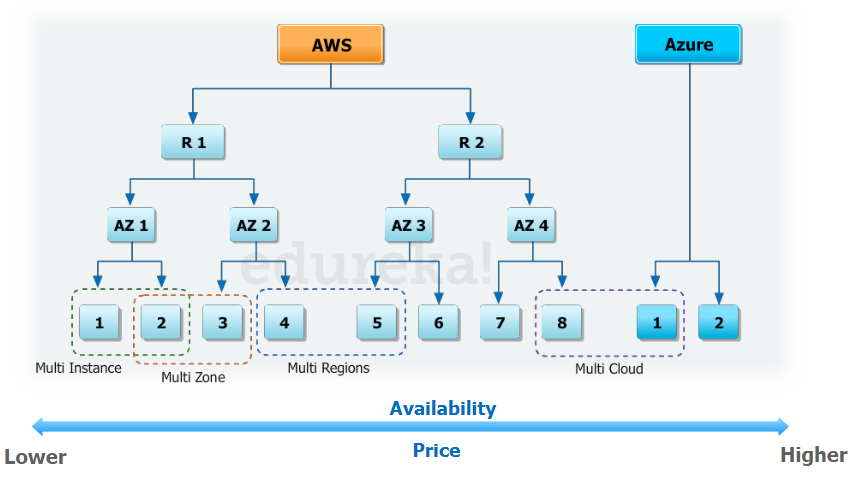
For more information, see [Creating a DB instance in a VPC](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_VPC.WorkingWithRDSInstanceinaVPC.html#USER_VPC.InstanceInVPC).

1. Create a DB instance that uses the DB subnet group in the Local Zone.

For more information, see [Creating an Amazon RDS DB instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html).

**Important**: Currently, the only AWS Local Zone where Amazon RDS is available is Los Angeles in the US West (Oregon) Region.

## High Availability (HA) with Multi-Region and Multi-AZ



# Create Windows EC2 Instance with Web Server

1. Create an internet facing instance in default VPC
2. Connect to the instance
3. Install IIS
   1. Server manager
   2. Add roles and features
   3. Role-based or feature-based installation
   4. Select the server
   5. Select Web Server (IIS)
   6. Features -> .NET Framework 4.8…
   7. Finish installation
4. Open Windows Explorer, go to C:\
5. Will create a new folder called “inetpub”
6. Go to C:\inetpub\wwwroot
7. Create an index.html file with text “This is a file on the web server!”
8. Get the public IP address of the instance
9. Navigate to the IP address in the browser
10. Displays the text
11. Modify the text file and then refresh the browser

# Create Custom AMI

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/creating-an-ami-ebs.html>

<https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/Creating_EBSbacked_WinAMI.html>

## Create an Amazon EBS-backed Linux AMI

The following diagram summarizes the process for creating an Amazon EBS-backed AMI from a running EC2 instance: Start with an existing AMI, launch an instance, customize it, create a new AMI from it, and finally launch an instance of your new AMI. The numbers in the diagram match the numbers in the description that follows.


    Workflow for creating an AMI from an instance
   

**1 – AMI #1: Start with an existing AMI**

Find an existing AMI that is similar to the AMI that you'd like to create. This can be an AMI you have obtained from the AWS Marketplace, an AMI you have created using the [AWS Server Migration Service](https://aws.amazon.com/server-migration-service/) or [VM Import/Export](https://docs.aws.amazon.com/vm-import/latest/userguide/what-is-vmimport.html), or any other AMI you can access. You'll customize this AMI for your needs.

In the diagram, **EBS root volume snapshot #1** indicates that the AMI is an Amazon EBS-backed AMI and that information about the root volume is stored in this snapshot.

**2 – Launch instance from existing AMI**

The way to configure an AMI is to launch an instance from the AMI on which you'd like to base your new AMI, and then customize the instance (indicated at **3** in the diagram). Then, you'll create a new AMI that includes the customizations (indicated at **4** in the diagram).

**3 – EC2 instance #1: Customize the instance**

Connect to your instance and customize it for your needs. Your new AMI will include these customizations.

You can perform any of the following actions on your instance to customize it:

* Install software and applications
* Copy data
* Reduce start time by deleting temporary files, defragmenting your hard drive, and zeroing out free space
* Attach additional EBS volumes

**4 – Create image**

When you create an AMI from an instance, Amazon EC2 powers down the instance before creating the AMI to ensure that everything on the instance is stopped and in a consistent state during the creation process. If you're confident that your instance is in a consistent state appropriate for AMI creation, you can tell Amazon EC2 not to power down and reboot the instance. Some file systems, such as XFS, can freeze and unfreeze activity, making it safe to create the image without rebooting the instance.

During the AMI-creation process, Amazon EC2 creates snapshots of your instance's root volume and any other EBS volumes attached to your instance. You're charged for the snapshots until you [deregister the AMI](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/deregister-ami.html) and delete the snapshots. If any volumes attached to the instance are encrypted, the new AMI only launches successfully on instances that support [Amazon EBS encryption](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html).

Depending on the size of the volumes, it can take several minutes for the AMI-creation process to complete (sometimes up to 24 hours). You might find it more efficient to create snapshots of your volumes before creating your AMI. This way, only small, incremental snapshots need to be created when the AMI is created, and the process completes more quickly (the total time for snapshot creation remains the same). For more information, see [Create Amazon EBS snapshots](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-creating-snapshot.html).

**5 – AMI #2: New AMI**

After the process completes, you have a new AMI and snapshot (**snapshot #2**) created from the root volume of the instance. If you added instance-store volumes or EBS volumes to the instance, in addition to the root device volume, the block device mapping for the new AMI contains information for these volumes.

Amazon EC2 automatically registers the AMI for you.

**6 – Launch instance from new AMI**

You can use the new AMI to launch an instance.

**7 – EC2 instance #2: New instance**

When you launch an instance using the new AMI, Amazon EC2 creates a new EBS volume for the instance's root volume using the snapshot. If you added instance-store volumes or EBS volumes when you customized the instance, the block device mapping for the new AMI contains information for these volumes, and the block device mappings for instances that you launch from the new AMI automatically contain information for these volumes. The instance-store volumes specified in the block device mapping for the new instance are new and don't contain any data from the instance store volumes of the instance you used to create the AMI. The data on EBS volumes persists. For more information, see [Block device mappings](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/block-device-mapping-concepts.html).

When you create a new instance from an EBS-backed AMI, you should initialize both its root volume and any additional EBS storage before putting it into production. For more information, see [Initialize Amazon EBS volumes](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-initialize.html).

**Steps**:

1. Create a new Linux instance
2. Select the instance -> Actions -> Images and Templates -> Create image
   1. For **Image name**, enter a unique name for the image, up to 127 characters.
   2. For **Image description**, enter an optional description of the image, up to 255 characters.
   3. For **No reboot**, either keep the **Enable** check box cleared (the default), or select it.

* If **Enable** is cleared, when Amazon EC2 creates the new AMI, it reboots the instance so that it can take snapshots of the attached volumes while data is at rest, in order to ensure a consistent state.
* If **Enable** is selected, when Amazon EC2 creates the new AMI, it does not shut down and reboot the instance.

**Warning**: If you choose to enable No reboot, we can't guarantee the file system integrity of the created image.

* 1. **Instance volumes** – You can modify the root volume, and add additional Amazon EBS and instance store volumes, as follows:
     1. The root volume is defined in the first row.
* To change the size of the root volume, for **Size**, enter the required value.
* If you select **Delete on termination**, when you terminate the instance created from this AMI, the EBS volume is deleted. If you clear **Delete on termination**, when you terminate the instance, the EBS volume is not deleted. For more information, see [Preserve Amazon EBS volumes on instance termination](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/terminating-instances.html#preserving-volumes-on-termination).
  + 1. To add an EBS volume, choose **Add volume** (which adds a new row). For **Storage type**, choose **EBS**, and fill in the fields in the row. When you launch an instance from your new AMI, additional volumes are automatically attached to the instance. Empty volumes must be formatted and mounted. Volumes based on a snapshot must be mounted.
    2. To add an instance store volume, see [Add instance store volumes to an AMI](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/add-instance-store-volumes.html#adding-instance-storage-ami). When you launch an instance from your new AMI, additional volumes are automatically initialized and mounted. These volumes do not contain data from the instance store volumes of the running instance on which you based your AMI.
  1. **Tags** – You can tag the AMI and the snapshots with the same tags, or you can tag them with different tags.
* To tag the AMI and the snapshots with the same tags, choose **Tag image and snapshots together**. The same tags are applied to the AMI and every snapshot that is created.
* To tag the AMI and the snapshots with different tags, choose **Tag image and snapshots separately**. Different tags are applied to the AMI and the snapshots that are created. However, all the snapshots get the same tags; you can't tag each snapshot with a different tag.

To add a tag, choose **Add tag**, and enter the key and value for the tag. Repeat for each tag.

* 1. When you're ready to create your AMI, choose **Create image.**

1. To view the status of your AMI while it is being created:
   1. In the navigation pane, choose **AMIs**.
   2. Set the filter to **Owned by me**, and find your AMI in the list.

Initially, the status is pending but should change to available after a few minutes.

## Launch an instance from an AMI you created

You can launch an instance from an AMI that you created from an instance or snapshot.

To launch an instance from your AMI:

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Images**, choose **AMIs**.
3. Set the filter to **Owned by me** and select your AMI.
4. Choose **Launch instance from AMI** (new console) or **Actions**, **Launch** (old console).
5. Accept the default values or specify custom values in the launch instance wizard.

## Install LAMP Web Server on Amazon Linux 2023

### Step 1: Prepare the LAMP server

1. [Connect to your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html#ec2-connect-to-instance-linux).
2. To ensure that all of your software packages are up to date, perform a quick software update on your instance. This process might take a few minutes, but it is important to make sure that you have the latest security updates and bug fixes.

The -y option installs the updates without asking for confirmation. If you would like to examine the updates before installing, you can omit this option.

[ec2-user ~]$ **sudo dnf update -y**

1. Install the latest versions of Apache web server and PHP packages for Amazon Linux 2023.

[ec2-user ~]$ **sudo dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel**

1. Install the MariaDB software packages. Use the **dnf install** command to install multiple software packages and all related dependencies at the same time.

[ec2-user ~]$ sudo dnf install mariadb105-server

You can view the current versions of these packages using the following command:

[ec2-user ~]$ sudo dnf info *package\_name*

Example:

[root@ip-172-31-25-170 ec2-user]# dnf info mariadb105

Last metadata expiration check: 0:00:16 ago on Tue Feb 14 21:35:13 2023.

Installed Packages

Name : mariadb105

Epoch : 3

Version : 10.5.16

Release : 1.amzn2023.0.6

Architecture : x86\_64

Size : 18 M

Source : mariadb105-10.5.16-1.amzn2023.0.6.src.rpm

Repository : @System

From repo : amazonlinux

Summary : A very fast and robust SQL database server

URL : http://mariadb.org

License : GPLv2 and LGPLv2

Description : MariaDB is a community developed fork from MySQL - a multi-user, multi-threaded

: SQL database server. It is a client/server implementation consisting of

: a server daemon (mariadbd) and many different client programs and libraries.

: The base package contains the standard MariaDB/MySQL client programs and

: utilities.

1. Start the Apache web server.

[ec2-user ~]$ **sudo systemctl start httpd**

1. Use the **systemctl** command to configure the Apache web server to start at each system boot.

[ec2-user ~]$ **sudo systemctl enable httpd**

You can verify that **httpd** is on by running the following command:

[ec2-user ~]$ **sudo systemctl is-enabled httpd**

1. Add a security rule to allow inbound HTTP (port 80) connections to your instance if you have not already done so. By default, a **launch-wizard-*N*** security group was created for your instance during launch. If you did not add additional security group rules, this group contains only a single rule to allow SSH connections.
   1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
   2. In the left navigator, choose **Instances**, and select your instance.
   3. On the **Security** tab, view the inbound rules. You should see the following rule:

Port range Protocol Source

22 tcp 0.0.0.0/0

**Warning**: Using 0.0.0.0/0 allows all IPv4 addresses to access your instance using SSH. This is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you authorize only a specific IP address or range of addresses to access your instance.

* 1. If there is no inbound rule to allow HTTP (port 80) connections, you must the add rule now. Choose the link for the security group. Using the procedures in [Add rules to a security group](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/working-with-security-groups.html#adding-security-group-rule), add a new inbound security rule with the following values:
* **Type**: HTTP
* **Protocol**: TCP
* **Port Range**: 80
* **Source**: Custom

1. Test your web server. In a web browser, type the public DNS address (or the public IP address) of your instance. If there is no content in /var/www/html, you should see the Apache test page, which will display the message "**It works!**".

You can get the public DNS for your instance using the Amazon EC2 console (check the **Public IPv4 DNS** column; if this column is hidden, choose **Preferences** (the gear-shaped icon) and toggle on **Public IPv4 DNS**).

Verify that the security group for the instance contains a rule to allow HTTP traffic on port 80. For more information, see [Add rules to a security group](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/working-with-security-groups.html#adding-security-group-rule).

**Important**: If you are not using Amazon Linux, you might also need to configure the firewall on your instance to allow these connections. For more information about how to configure the firewall, see the documentation for your specific distribution.

Apache **httpd** serves files that are kept in a directory called the Apache document root. The Amazon Linux Apache document root is /var/www/html, which by default is owned by root.

To allow the ec2-user account to manipulate files in this directory, you must modify the ownership and permissions of the directory. There are many ways to accomplish this task. In this tutorial, you add ec2-user to the apache group to give the apache group ownership of the /var/www directory and assign write permissions to the group.

**To set file permissions**

1. Add your user (in this case, ec2-user) to the apache group.

[ec2-user ~]$ **sudo usermod -a -G apache *ec2-user***

1. Log out and then log back in again to pick up the new group, and then verify your membership.
   1. Log out (use the **exit** command or close the terminal window):

[ec2-user ~]$ **exit**

* 1. To verify your membership in the apache group, reconnect to your instance, and then run the following command:

[ec2-user ~]$ **groups**

ec2-user adm wheel apache systemd-journal

1. Change the group ownership of /var/www and its contents to the apache group.

[ec2-user ~]$ **sudo chown -R ec2-user:apache /var/www**

1. To add group write permissions and to set the group ID on future subdirectories, change the directory permissions of /var/www and its subdirectories.

[ec2-user ~]$ **sudo chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;**

1. To add group write permissions, recursively change the file permissions of /var/www and its subdirectories:

[ec2-user ~]$ **find /var/www -type f -exec sudo chmod 0664 {} \;**

Now, ec2-user (and any future members of the apache group) can add, delete, and edit files in the Apache document root, enabling you to add content, such as a static website or a PHP application.

**To secure your web server (Optional)**

A web server running the HTTP protocol provides no transport security for the data that it sends or receives. When you connect to an HTTP server using a web browser, the URLs that you visit, the content of webpages that you receive, and the contents (including passwords) of any HTML forms that you submit are all visible to eavesdroppers anywhere along the network pathway. The best practice for securing your web server is to install support for HTTPS (HTTP Secure), which protects your data with SSL/TLS encryption.

### Step 2: Test your LAMP server

If your server is installed and running, and your file permissions are set correctly, your ec2-user account should be able to create a PHP file in the /var/www/html directory that is available from the internet.

**To test your LAMP server**

1. Create a PHP file in the Apache document root.

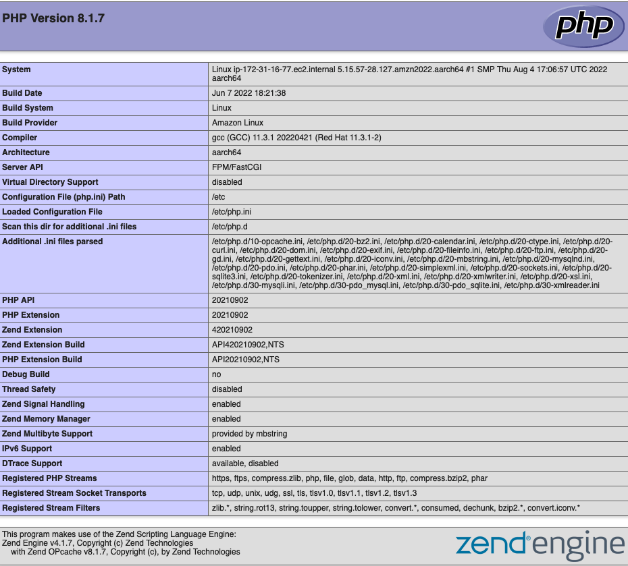
[ec2-user ~]$ **echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php**

If you get a "Permission denied" error when trying to run this command, try logging out and logging back in again to pick up the proper group permissions that you configured in [To set file permissions](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-lamp-amazon-linux-2023.html#setting-file-permissions-2023).

1. In a web browser, type the URL of the file that you just created. This URL is the public DNS address of your instance followed by a forward slash and the file name. For example:

http://*my.public.dns.amazonaws.com*/phpinfo.php

You should see the PHP information page:



If you do not see this page, verify that the /var/www/html/phpinfo.php file was created properly in the previous step. You can also verify that all of the required packages were installed with the following command.

[ec2-user ~]$ **sudo dnf list installed httpd mariadb-server php-mysqlnd**

If any of the required packages are not listed in your output, install them with the **sudo yum install *package*** command.

1. Delete the phpinfo.php file. Although this can be useful information, it should not be broadcast to the internet for security reasons.

[ec2-user ~]$ **rm /var/www/html/phpinfo.php**

You should now have a fully functional LAMP web server. If you add content to the Apache document root at /var/www/html, you should be able to view that content at the public DNS address for your instance.

### Step 3: Secure the database server (optional)

The default installation of the MariaDB server has several features that are great for testing and development, but they should be disabled or removed for production servers. The **mysql\_secure\_installation** command walks you through the process of setting a root password and removing the insecure features from your installation. Even if you are not planning on using the MariaDB server, we recommend performing this procedure.

**To secure the MariaDB server**

1. Start the MariaDB server.

[ec2-user ~]$ **sudo systemctl start mariadb**

1. Run mysql\_secure\_installation.

[ec2-user ~]$ **sudo mysql\_secure\_installation**

* 1. When prompted, type a password for the root account.
     1. Type the current root password. By default, the root account does not have a password set. Press Enter.
     2. Type **Y** to set a password, and type a secure password twice. For more information about creating a secure password, see <https://identitysafe.norton.com/password-generator/>. Make sure to store this password in a safe place.

Setting a root password for MariaDB is only the most basic measure for securing your database. When you build or install a database-driven application, you typically create a database service user for that application and avoid using the root account for anything but database administration.

* 1. Type Y to remove the anonymous user accounts.
  2. Type Y to disable the remote root login.
  3. Type Y to remove the test database.
  4. Type Y to reload the privilege tables and save your changes.

1. (Optional) If you do not plan to use the MariaDB server right away, stop it. You can restart it when you need it again.

[ec2-user ~]$ **sudo systemctl stop mariadb**

1. (Optional) If you want the MariaDB server to start at every boot, type the following command.

[ec2-user ~]$ **sudo systemctl enable mariadb**

### Step 4: (Optional) Install phpMyAdmin

[phpMyAdmin](https://www.phpmyadmin.net/) is a web-based database management tool that you can use to view and edit the MySQL databases on your EC2 instance. Follow the steps below to install and configure phpMyAdmin on your Amazon Linux instance.

**Important**: We do not recommend using phpMyAdmin to access a LAMP server unless you have enabled SSL/TLS in Apache; otherwise, your database administrator password and other data are transmitted insecurely across the internet. For security recommendations from the developers, see [Securing your phpMyAdmin installation](https://docs.phpmyadmin.net/en/latest/setup.html#securing-your-phpmyadmin-installation). For general information about securing a web server on an EC2 instance, see [Tutorial: Configure SSL/TLS on Amazon Linux 2023](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2023.html).

**To install phpMyAdmin**

1. Install the required dependencies.

[ec2-user ~]$ **sudo dnf install php-mbstring php-xml -y**

1. Restart Apache.

[ec2-user ~]$ **sudo systemctl restart httpd**

1. Restart php-fpm.

[ec2-user ~]$ **sudo systemctl restart php-fpm**

1. Navigate to the Apache document root at /var/www/html.

[ec2-user ~]$ **cd /var/www/html**

1. Select a source package for the latest phpMyAdmin release from <https://www.phpmyadmin.net/downloads>. To download the file directly to your instance, copy the link and paste it into a **wget** command, as in this example:

[ec2-user html]$ **wget *https://www.phpmyadmin.net/downloads/phpMyAdmin-latest-all-languages.tar.gz***

1. Create a phpMyAdmin folder and extract the package into it with the following command.

[ec2-user html]$ **mkdir phpMyAdmin && tar -xvzf *phpMyAdmin-latest-all-languages.tar.gz* -C phpMyAdmin --strip-components 1**

1. Delete the *phpMyAdmin-latest-all-languages.tar.gz* tarball.

[ec2-user html]$ **rm *phpMyAdmin-latest-all-languages.tar.gz***

1. (Optional) If the MySQL server is not running, start it now.

[ec2-user ~]$ **sudo systemctl start mariadb**

1. In a web browser, type the URL of your phpMyAdmin installation. This URL is the public DNS address (or the public IP address) of your instance followed by a forward slash and the name of your installation directory. For example:

http://*my.public.dns.amazonaws.com*/phpMyAdmin

You should see the phpMyAdmin login page:



1. Log in to your phpMyAdmin installation with the root user name and the MySQL root password you created earlier.

Your installation must still be configured before you put it into service. We suggest that you begin by manually creating the configuration file, as follows:

* 1. To start with a minimal configuration file, use your favorite text editor to create a new file, and then copy the contents of config.sample.inc.php into it.
  2. Save the file as config.inc.php in the phpMyAdmin directory that contains index.php.
  3. Refer to post-file creation instructions in the [Using the Setup script](https://docs.phpmyadmin.net/en/latest/setup.html#using-the-setup-script) section of the phpMyAdmin installation instructions for any additional setup.

## Create AMI from LAMP server instance

1. Select the instance -> Actions -> Images and Templates -> Create image
2. In the navigation pane, under **Images**, choose **AMIs**.
3. Set the filter to **Owned by me** and select your AMI.
4. Choose **Launch instance from AMI** (new console) or **Actions**, **Launch** (old console).
5. Accept the default values or specify custom values in the launch instance wizard.

# Amazon Identity and Access Management (IAM)

## IAM Roles

<https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html>

An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

You can use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources. For example, you might want to grant users in your AWS account access to resources they don't usually have, or grant users in one AWS account access to resources in another account. Or you might want to allow a mobile app to use AWS resources, but not want to embed AWS keys within the app (where they can be difficult to rotate and where users can potentially extract them). Sometimes you want to give AWS access to users who already have identities defined outside of AWS, such as in your corporate directory. Or, you might want to grant access to your account to third parties so that they can perform an audit on your resources.

An IAM identity that you can create in your account that has specific permissions. An IAM role has some similarities to an IAM user. Roles and users are both AWS identities with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

Roles can be used by the following:

* An IAM user in the same AWS account as the role
* An IAM user in a different AWS account than the role
* A web service offered by AWS such as Amazon Elastic Compute Cloud (Amazon EC2)
* An external user authenticated by an external identity provider (IdP) service that is compatible with SAML 2.0 or OpenID Connect, or a custom-built identity broker.

## Common scenarios for roles: Users, applications, and services

<https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios.html>

As with most AWS features, you generally have two ways to use a role: interactively in the IAM console, or programmatically with the AWS CLI, Tools for Windows PowerShell, or API.

* IAM users in your account using the IAM console can switch to a role to temporarily use the permissions of the role in the console. The users give up their original permissions and take on the permissions assigned to the role. When the users exit the role, their original permissions are restored.
* An application or a service offered by AWS (like Amazon EC2) can assume a role by requesting temporary security credentials for a role with which to make programmatic requests to AWS. You use a role this way so that you don't have to share or maintain long-term security credentials (for example, by creating an IAM user) for each entity that requires access to a resource.

**Note**: This guide uses the phrases *switch to a role* and *assume a role* interchangeably.

## Providing access to an AWS service

<https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios_services.html>

Many AWS services require that you use roles to control what that service can access. A role that a service assumes to perform actions on your behalf is called a [service role](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html#iam-term-service-role). When a role serves a specialized purpose for a service, it can be categorized as a [service role for EC2 instances](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html#iam-term-service-role-ec2), or a [service-linked role](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html#iam-term-service-linked-role). See the [AWS documentation](https://docs.aws.amazon.com/) for each service to see if it uses roles and to learn how to assign a role for the service to use.

For details about creating a role to delegate access to a service offered by AWS, see [Creating a role to delegate permissions to an AWS service](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-service.html).

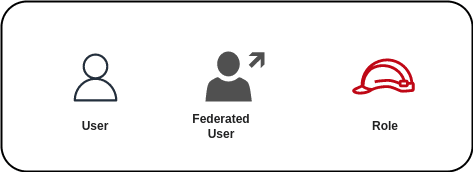
## Identities: Roles, Users and Federated Identities

<https://spacelift.io/blog/aws-iam-roles>

Identities on AWS are the way to identify principals. A principal can be a person or an app that performs actions in the AWS account.

Identities can be classified into 3 major categories:

* AWS users
* Federated identities
* AWS roles

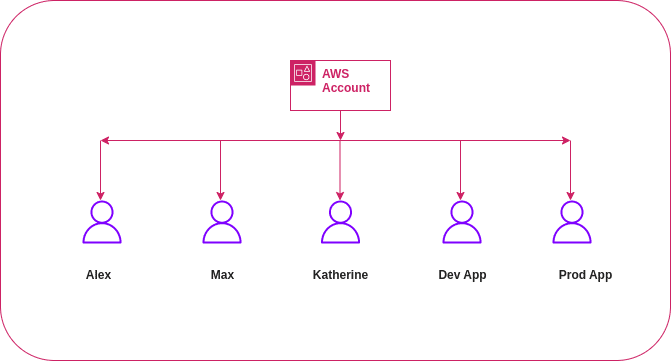


### AWS Users

AWS users are principals (persons or apps) with their own *usernames and passwords* to allow access to the AWS console.

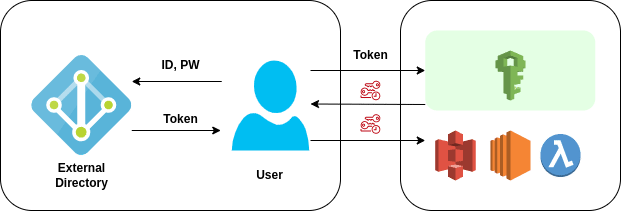
We can also *generate access keys* that are tied to a particular user that can be used with the AWS CLI or SDKs.

**Note**: Notice that some of the users listed below are actually applications. As previously stated, an AWS user does not have to be a real person; it can also be an app with its own set of access keys.



### Federated Identities

These are users managed outside of AWS. Identity providers(IdP) can be used to grant these users access to use AWS resources without having to create users within the AWS account. AWS supports OIDC and SAML 2.0 compatible IdPs.



### IAM Roles

IAM roles are particularly intriguing because they are conceptually similar to AWS users but differ in the way that a user is uniquely associated with a principal(users/apps/etc.) whereas a role is not and can be assumed by anyone who requires it.

Roles do not have passwords or access keys associated with them. Instead, roles provide temporary security credentials to whoever assumes the role.

This is important considering that the *users having access to your AWS accounts can change over time but the roles used to manage your AWS account don’t change often*. Roles eliminate the overhead of managing users and their long-lived credentials by generating temporary credentials whenever required.

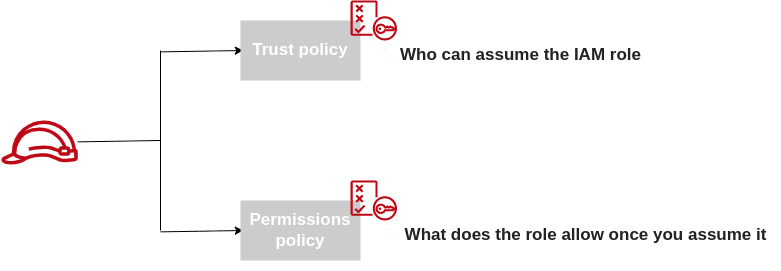
In fact, AWS recommends using temporary IAM roles over IAM users.

## Structure of an IAM role

From the above definition, it can be inferred that there are 2 essential aspects to an IAM role:

1. **Trust policy**: Who can assume the IAM role
2. **Permission policy**: What does the role allow once you assume it

Let us break them down to understand what makes up an IAM role.



Let us define these before we move on to creating our first AWS role.

### Trust policy

These are policies that define and control which principals can assume the role based on the specified conditions. Trust policies are used to prevent the misuse of IAM roles by unauthorized or unintended entities

### **Permission policy**

These policies define what the principals who assume the role are allowed to do once they assume it.

With the fundamentals of an IAM role established, we can proceed to create our first IAM role.

## **Creating Our First Role**

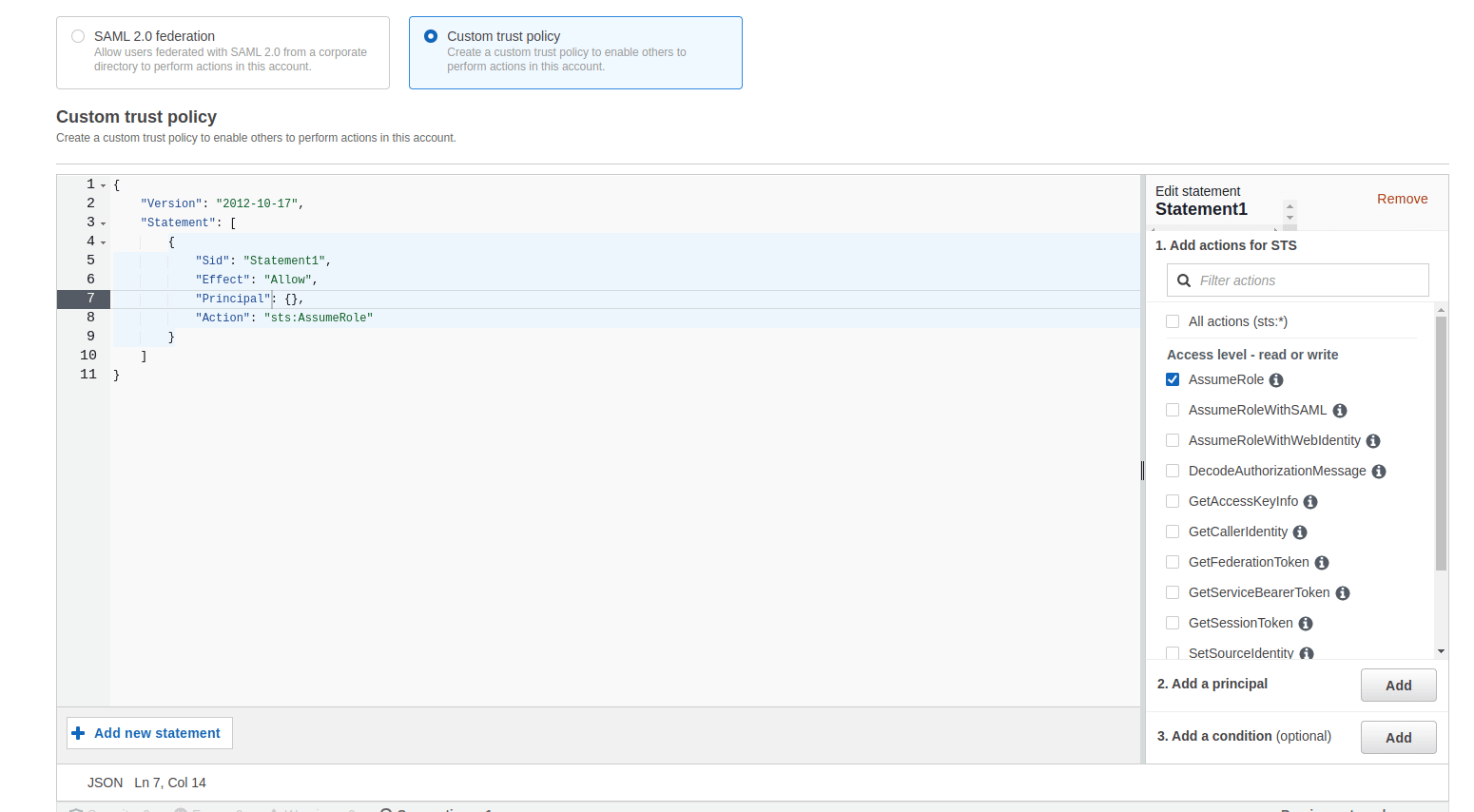
### **Creating our first role**

**Pre-requisite**: An AWS account with an IAM admin user.

In this tutorial, we will create a custom IAM role with the trusted entity as the AWS user and the permission policy to allow admin access to [AWS RDS](https://aws.amazon.com/rds/).



1. Log in to the AWS account and open the IAM service.
2. Open the roles panel either from the dashboard or from the side nav on the left.
3. You might notice some pre-created roles in your account on the roles panel. Ignore them for the time being; we will go over them in greater detail later in the article.
4. To create a new role, click on the “Create role” button.
5. As previously stated, a role has 2 core aspects, ***the trust policy*** and ***the permission policy***. So the first thing that we have to specify is who can assume this role. We will begin by selecting the “Custom trust policy” and then proceed to the other options in the later section of the article.
6. Upon selection of the “Custom trust policy”, AWS automatically generates a JSON policy with an empty principal field. The principal field specifies who can assume this role. If we keep it empty then this policy cannot be assumed by any principal.



1. We will add the [Amazon Resource Name](https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html) (ARN) of the AWS IAM user who should be allowed to assume this role. The ARN can be obtained from the user details page in the IAM dashboard.
2. Copy the ARN and paste it as a key-value pair, with the key being “AWS” as shown below.

{

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

"Statement": [

{

"Sid": "Statement1",

"Effect": "Allow",

"Principal": {

"AWS": "arn:aws:iam::xxxxxxxxxxx:user/omkar"

},

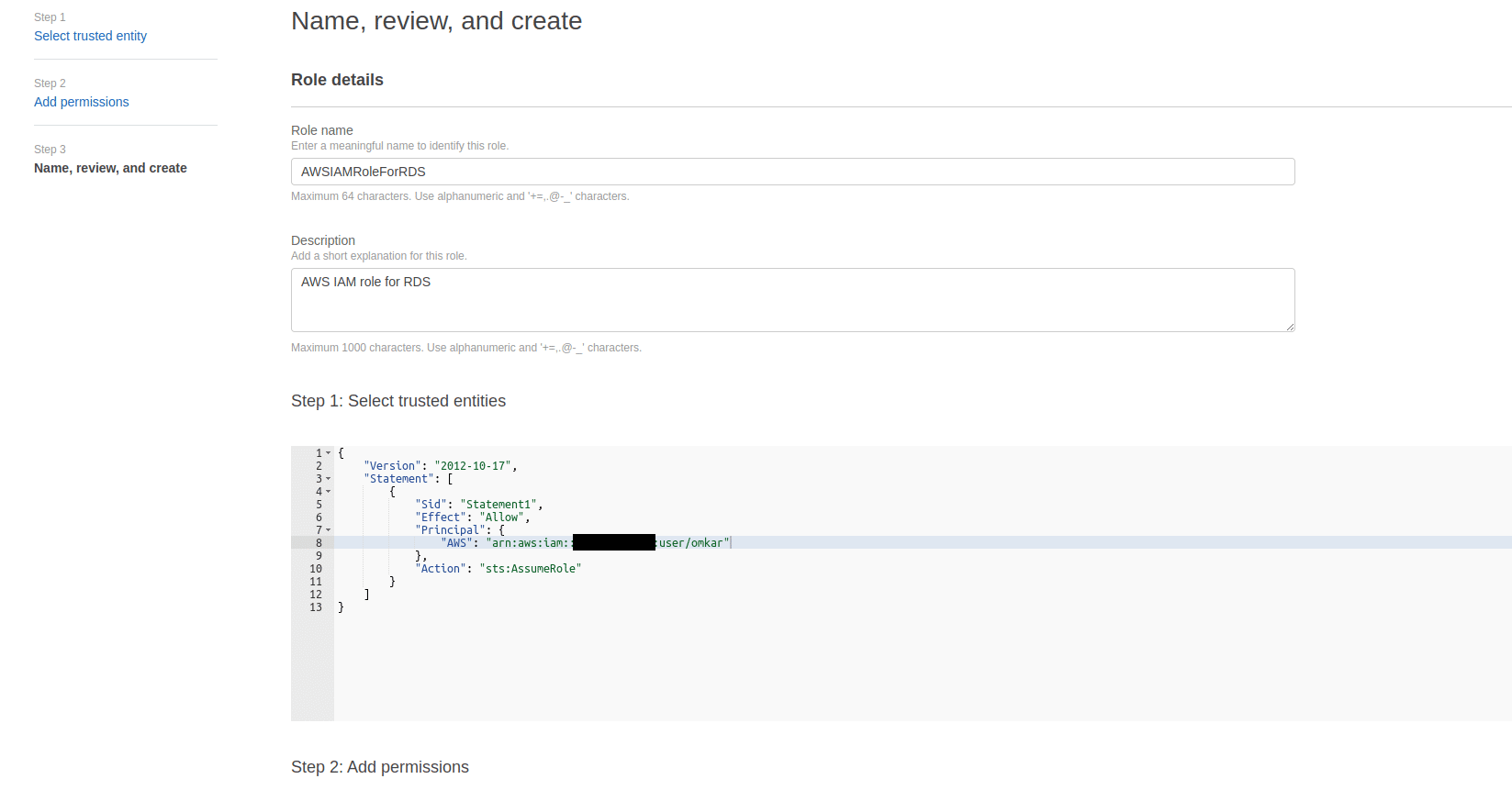
"Action": "sts:AssumeRole"

}

]

}

1. Once you have reviewed the trust policy, click on the next button to move on to the next page.
2. The next step, as you might expect, is to choose a *permission policy* for this role. We can either use an existing policy or create a new one. We will choose an existing managed policy by the name *AmazonRDSFullAccess* to grant our role full access to the AWS RDS service.
3. **Remember that AWS denies everything by default.** We are only granting the RDS access to this role by attaching this policy.
4. Leave all the other settings unchanged and click on next.
5. We have already taken care of the 2 essential aspects of a role. All that is left is to give the role a name and a description, and we are done.



1. Name the role as AWSIAMRoleForRDS and provide the description as “AWS IAM role for RDS”
2. Review all the details and click on the **Create role** button.

Roles, as previously stated, do not have any long-lived credentials associated with them, but they can be assumed by identities via the console, CLI, or API. In the following section, we will learn how to assume an IAM role.

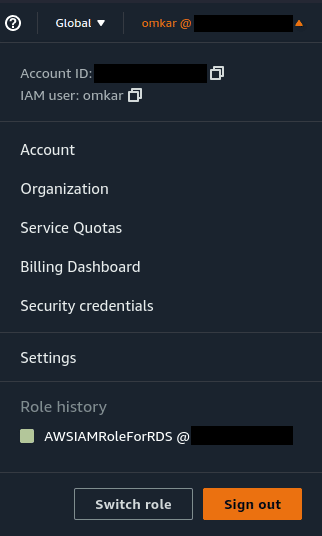
### Assuming IAM Roles

There are multiple ways to assume IAM roles. We can assume a role using the console, the CLI or even the SDK. Let’s go through all of them one by one.

#### Switching to a role through the console

To switch to a role using the console, you have to be logged in as an IAM user allowed to assume that role.

1. We can switch to a different role using the user information panel in the top right corner. When we click on it, the “Switch role” button appears next to the “Sign out” button.



1. Upon clicking on the “Switch role” button, we are greeted by a form requesting information about the role we wish to switch to.

If you are switching your role for the first time you would be greeted by a different screen explaining the details about switching a role. To proceed, click on the “Switch role” button.

1. Fill in the role details with the role we created in the previous tutorial and click on the “Switch role” button.

Upon switching to the IAM role, we arrive at the IAM dashboard again with the principal as the AWSIAMRoleForRDS@1234567890.

The interesting thing to note here is the big error message that pops up as soon as we switch to this role stating:

User: arn:aws:sts::1234567890:assumed-role/AWSIAMRoleForRDS/<username> is not authorized to perform: iam:ListRoles on resource: arm:aws:iam::123456789:role/ because no identity-based policy allows the iam:ListRoles action

Do you know why this error has appeared?

Remember that when creating the permission policy for our role, we only selected the AWS RDS access and nothing else. This means that the assumed role is only allowed access to AWS RDS and we are currently on the IAM dashboard.

**Note:**It is important to remember that when we assume a role we inherit the role’s permission but at the same time we end up setting aside the permissions assigned to our original user which we used to assume the role.

Let us take our newly created and assumed IAM role on a pilot run and try to create a database instance.

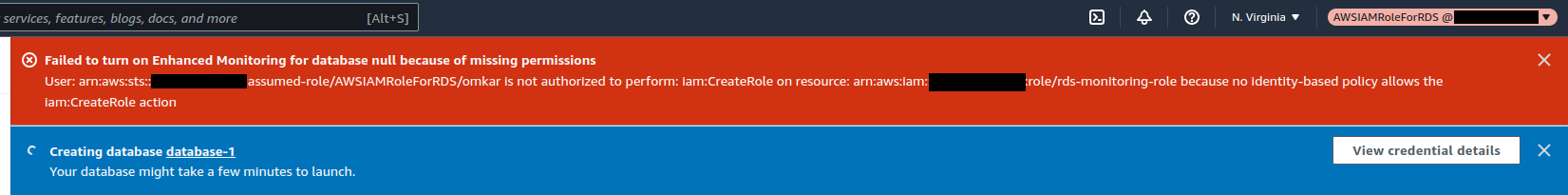
### **Using an IAM role to create resources**

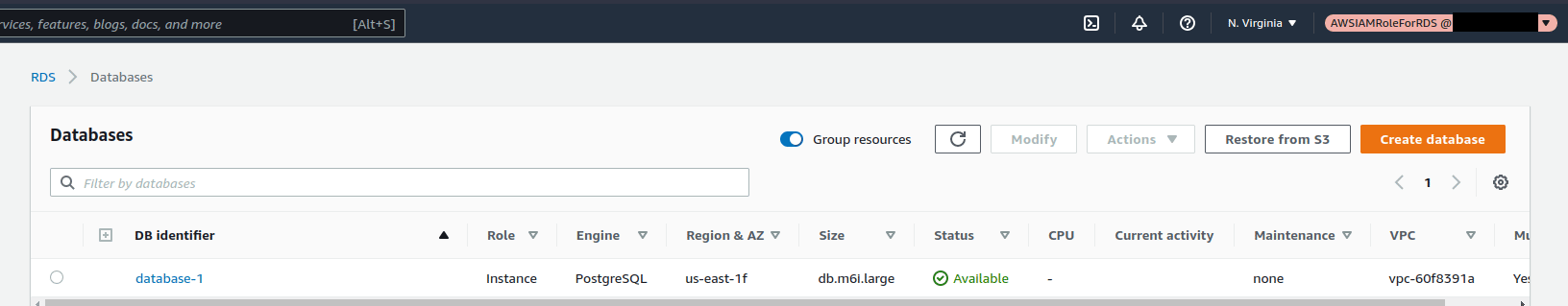
We granted our IAM role complete access to AWS RDS. This means that it has read, written as well as created permission for the RDS service. Before proceeding, please make sure that you have already assumed the AWSRDSIAMRole role

1. Go to the AWS RDS service dashboard. If things are configured right, we would not see any unauthorized errors.
2. Next, we will try to spin up a database instance. Please refer to [this](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.PostgreSQL.html) tutorial to create a database instance.

**Note 1:**Make sure to uncheck the performance insights checkbox to avoid creating a KMS key.

**Note 2:**A few unauthorized errors that may appear for the creation of resources required for monitoring, etc., are worth noting here, as the permissions of our role are only limited to RDS. However, because the majority of these steps are not mandatory for the creation of a database, it will continue without stopping.





1. Awesome, you successfully created your first resource using the assumed role.

### **To stop using a role (AWS Console)**

To stop using the assumed role, simply click on the **Switch back** button in the user details panel in the top right corner.

# Amazon Simple Storage Service (S3)

## Create a Static Web Site

1. In the AWS management console, search for the S3 service and select it
2. Click on the create bucket
3. Now specify the name and region of your bucket
4. Click on create
5. Select the bucket -> Properties tab
6. Towards the end of the page click on Edit for `Static website hosting`.
7. Select Enable for `Static website hosting`. For the `Index document` enter `index.html.
8. Save changes
9. Note down the **URL** at the end under “Static website hosting”. We would be using this to access the web pages in S3 via the browser later.
10. Enable Use this bucket to host a website and index document as index.html
11. In permission tab, select Bucket Policy

**S3 Static Web Site\BucketPolicy.json**

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

"Statement": [

{ "Sid": "AddPerm",

"Effect": "Allow",

"Principal": "\*",

"Action": [

"s3:GetObject"

],

"Resource": [ "arn:aws:s3:::edureka-training/\*" //give your bucket name ]

}

]

}

1. Type the above bucket policy and save it.
2. Create an index.html page on your local system
3. Add the below code into index.html

**S3 Static Web Site\index.html**:

<!DOCTYPE html>

<html>

<head>

<style>

.a{

background-color: #2471A3;

color: white;

padding: 12px 20px;

border: none;

border-radius: 4px;

cursor: pointer;

float: center;

}

.bg {

/\* The image used \*/

background-image: url("https://bit.ly/2OEVTYp");

/\* Full height \*/

height: 100%;

/\* Center and scale the image nicely \*/

background-position: center;

background-repeat: no-repeat;

background-size: cover;

}

.label {

color: white;

padding: 8px;

font-family: Arial; }

}

</style>

</head>

<body class="bg" style="padding: 210px 0; background-color: #dbfcf9;">

<center>

<h3><font size="24"> <font color="white">S3 Demo </font></h3>

</center>

</body>

</html>

1. Upload index.html to your bucket
2. Access the url of the site

## S3 Bucket Policy

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-bucket-policies.html>

With Amazon S3 bucket policies, you can secure access to objects in your buckets, so that only users with the appropriate permissions can access them. You can even prevent authenticated users without the appropriate permissions from accessing your Amazon S3 resources.

To grant or deny permissions to a set of objects, you can use wildcard characters (\*) in Amazon Resource Names (ARNs) and other values. For example, you can control access to groups of objects that begin with a common [prefix](https://docs.aws.amazon.com/general/latest/gr/glos-chap.html#keyprefix) or end with a given extension, such as .html.

### Getting a List of Objects in a Bucket with a Specific Prefix

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazon-s3-policy-keys.html#example-object-tagging-access-control>

You can use the s3:prefix condition key to limit the response of the [GET Bucket (ListObjects)](https://docs.aws.amazon.com/AmazonS3/latest/API/API_ListObjects.html) API to key names with a specific prefix. If you are the bucket owner, you can restrict a user to list the contents of a specific prefix in the bucket. This condition key is useful if objects in the bucket are organized by key name prefixes. The Amazon S3 console uses key name prefixes to show a folder concept. Only the console supports the concept of folders; the Amazon S3 API supports only buckets and objects. For more information about using prefixes and delimiters to filter access permissions, see [Controlling access to a bucket with user policies](https://docs.aws.amazon.com/AmazonS3/latest/userguide/walkthrough1.html).

For example, if you have two objects with key names public/object1.jpg and public/object2.jpg, the console shows the objects under the public folder. In the Amazon S3 API, these are objects with prefixes, not objects in folders. However, in the Amazon S3 API, if you organize your object keys using such prefixes, you can grant s3:ListBucket permission with the s3:prefix condition that will allow the user to get a list of key names with those specific prefixes.

In this example, the bucket owner and the parent account to which the user belongs are the same. So the bucket owner can use either a bucket policy or a user policy. For more information about other condition keys that you can use with the GET Bucket (ListObjects) API, see [ListObjects](https://docs.aws.amazon.com/AmazonS3/latest/API/API_ListObjects.html).

{

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

"Statement":[

{

"Sid":"statement1",

"Effect":"Allow",

"Principal": {

"AWS": "arn:aws:iam::*123456789012*:user/*bucket-owner*"

},

"Action": "s3:ListBucket",

"Resource": "arn:aws:s3:::*awsexamplebucket1*",

"Condition" : {

"StringEquals" : {

"s3:prefix": "projects"

}

}

},

{

"Sid":"statement2",

"Effect":"Deny",

"Principal": {

"AWS": "arn:aws:iam::*123456789012*:user/*bucket-owner*"

},

"Action": "s3:ListBucket",

"Resource": "arn:aws:s3:::*awsexamplebucket1*",

"Condition" : {

"StringNotEquals" : {

"s3:prefix": "projects"

}

}

}

]

}

#### Test the policy with the AWS CLI

You can test the policy using the following list-object AWS CLI command. In the command, you provide user credentials using the --profile parameter. For more information about setting up and using the AWS CLI, see [Developing with Amazon S3 using the AWS CLI](https://docs.aws.amazon.com/AmazonS3/latest/userguide/setup-aws-cli.html).

aws s3api list-objects --bucket *awsexamplebucket1* --prefix examplefolder --profile AccountADave

aws s3 ls *awsexamplebucket1* /projects --profile AccountADave

If the bucket is version-enabled, to list the objects in the bucket, you must grant the s3:ListBucketVersions permission in the preceding policy, instead of s3:ListBucket permission. This permission also supports the s3:prefix condition key.

### Allow a User to Read Only Objects That Have a Specific Tag Key and Value

The following permissions policy limits a user to only reading objects that have the environment: production tag key and value. This policy uses the s3:ExistingObjectTag condition key to specify the tag key and value.

{

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

"Statement":[

{

"Principal":{

"AWS":"arn:aws:iam::111122223333:role/JohnDoe"

},

"Effect":"Deny",

"Action":[

"s3:GetObject",

"s3:GetObjectVersion"

],

"Resource":"arn:aws:s3:::DOC-EXAMPLE-BUCKET/\*",

"Condition":{

"StringNotEquals":{

"s3:ExistingObjectTag/environment":"production"

}

}

}

]

}

#### Test the policy with the AWS CLI

You can test the policy using the following list-object AWS CLI command. In the command, you provide user credentials using the --profile parameter. For more information about setting up and using the AWS CLI, see [Developing with Amazon S3 using the AWS CLI](https://docs.aws.amazon.com/AmazonS3/latest/userguide/setup-aws-cli.html).

aws s3api get-object --bucket *awsexamplebucket1* --key src .\target

aws s3 cp s3://*awsexamplebucket1*/src .\target

### Allow a User to Only Add Objects with a Specific Object Tag Key and Value

The following example policy grants a user permission to perform the s3:PutObject action so that they can add objects to a bucket. However, the Condition statement restricts the tag keys and values that are allowed on the uploaded objects. In this example, the user can only add objects that have the specific tag key (*Department*) with the value set to *Finance* to the bucket.

{

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

"Statement": [{

"Principal":{

"AWS":[

"arn:aws:iam::*111122223333*:user/*JohnDoe*"

]

},

"Effect": "Deny",

"Action": [

"s3:PutObject"

],

"Resource": [

"arn:aws:s3:::*DOC-EXAMPLE-BUCKET*/\*"

],

"Condition": {

"StringNotEquals": {

"s3:RequestObjectTag/*Department*": "*Finance*"

}

}

}]

}

#### Test the policy with the AWS CLI

You can test the policy using the following list-object AWS CLI command. In the command, you provide user credentials using the --profile parameter. For more information about setting up and using the AWS CLI, see [Developing with Amazon S3 using the AWS CLI](https://docs.aws.amazon.com/AmazonS3/latest/userguide/setup-aws-cli.html).

aws s3api put-object --bucket *awsexamplebucket1*--key target --body .\src --tagging Department=Finance

### Restrict Access to Specific IP Addresses

The following example denies all users from performing any Amazon S3 operations on objects in the specified buckets unless the request originates from the specified range of IP addresses.

This policy's Condition statement identifies *192.0.2.0/24* as the range of allowed Internet Protocol version 4 (IPv4) IP addresses.

The Condition block uses the NotIpAddress condition and the aws:SourceIp condition key, which is an AWS wide condition key. The aws:SourceIp condition key can only be used for public IP address ranges. For more information about these condition keys, see [Amazon S3 condition key examples](https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazon-s3-policy-keys.html). The aws:SourceIp IPv4 values use standard CIDR notation. For more information, see [IAM JSON Policy Elements Reference](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html#Conditions_IPAddress) in the IAM User Guide.

**Warning**: Before using this policy, replace the *192.0.2.0/24* IP address range in this example with an appropriate value for your use case. Otherwise, you will lose the ability to access your bucket.

{

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

"Id": "S3PolicyId1",

"Statement": [

{

"Sid": "IPAllow",

"Effect": "Deny",

"Principal": "\*",

"Action": "s3:\*",

"Resource": [

"arn:aws:s3:::*DOC-EXAMPLE-BUCKET*",

"arn:aws:s3:::*DOC-EXAMPLE-BUCKET1*/\*"

],

"Condition": {

"NotIpAddress": {

"aws:SourceIp": "*192.0.2.0/24*"

}

}

}

]

}

# Create VPC with IGW and NGW

1. Create a VPC: 10.0.0.0/16
2. Create a public subnet: 10.0.1.0/24 in AZ-a
3. Create a private subnet: 10.0.2.0/24 in AZ-b
4. Create an Internet Gateway (IGW)
5. Create a Route Table for public subnet
   1. Associate to VPC
   2. Associate public subnet
   3. Add routes: 0.0.0.0/0 and select our IGW.
6. Create an EC2 instance – Window – Pubic subnet
   1. Windows
   2. Key pair
   3. Our VPC
   4. Public subnet
   5. Auto-assign public IP - Yes
   6. Security Group Inbound rules:
      1. RDP - TCP – 3389 – Anywhere-IPv4
      2. HTTP - HTTP – 80– Anywhere-IPv4
      3. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
   7. Launch
7. Connect to instance
   1. Download RDP file
   2. Get password from .pem file (of key pair)
   3. Username: Administrator
   4. Browser check
8. Create NAT Gateway (NGW)
   1. Select public subnet
   2. Connectivity type: Public
   3. Attach Elastic IP
9. Create a Route Table for private subnet
   1. Associate to VPC
   2. Associate private subnet
   3. Add routes: 0.0.0.0/0 and select our NGW.
10. Create an EC2 instance – Window – Private subnet
    1. Windows
    2. Key pair
    3. Our VPC
    4. Private subnet
    5. Auto-assign public IP - No
    6. Security Group Inbound rules: Use same SG created with public instance
       1. RDP - TCP – 3389 – Anywhere-IPv4
       2. HTTP - HTTP – 80– Anywhere-IPv4
       3. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    7. Launch
11. Connect to instance using private IP
    1. Download RDP file
    2. Get password from .pem file (of key pair)
    3. Username: Administrator
    4. Browser check

# Create VPCs Peering Connection

*AWS Solution Architect – Class 6 recording*

## Create VPC, Subnets and Instance in US

1. Change Region to US-East N. Virginia on top-right.
2. Create a VPC: 10.0.0.0/16 in Region-A (US).
3. Create a public subnet: 10.0.1.0/24 in AZ-a
4. Create a private subnet: 10.0.2.0/24 in AZ-a
5. Create an Internet Gateway (IGW) – Name: US-IGW
6. Create a Route Table for public subnet
   1. Associate to VPC (US)
   2. Associate public subnet
   3. Add routes: 0.0.0.0/0 and select our US-IGW.
7. Create an EC2 instance – Window – Pubic subnet
   1. Windows
   2. Create New Key pair
   3. Our VPC (US)
   4. Public subnet
   5. Auto-assign public IP - Yes
   6. Security Group Inbound rules:
      1. RDP - TCP – 3389 – Anywhere-IPv4
      2. HTTP - HTTP – 80– Anywhere-IPv4
      3. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
   7. Launch
8. Connect to instance
   1. Download RDP file
   2. Get password from .pem file (of key pair)
   3. Username: Administrator
   4. Browser check
9. Create NAT Gateway (NGW) – Name: US-NGW
   1. Select public subnet
   2. Connectivity type: Public
   3. Attach Elastic IP
10. Create a Route Table for private subnet
    1. Associate to VPC (US)
    2. Associate private subnet
    3. Add routes: 0.0.0.0/0 and select our US-NGW.
11. Create an EC2 instance – Window – Private subnet
    1. Windows
    2. Use same Key pair
    3. Our VPC (US)
    4. Private subnet
    5. Auto-assign public IP - No
    6. Security Group Inbound rules: Use same SG created with public instance
       1. RDP - TCP – 3389 – Anywhere-IPv4
       2. HTTP - HTTP – 80– Anywhere-IPv4
       3. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    7. Launch
12. Connect to instance using private IP
    1. Download RDP file
    2. Get password from .pem file (of key pair)
    3. Username: Administrator
    4. Browser check

## Create VPC, Subnets and Instance in Europe (EU)

1. Change Region to London on top-right.
2. Create a VPC: 192.168.0.0/16 in Region-A (EU).
3. Create a public subnet: 192.168.1.0/24 in AZ-a
4. Create a private subnet: 192.168.2.0/24 in AZ-a
5. Create an Internet Gateway (IGW) – Name: EU-IGW
6. Create a Route Table for public subnet
   1. Associate to VPC (EU)
   2. Associate public subnet
   3. Add routes: 0.0.0.0/0 and select our EU-IGW.
7. Create an EC2 instance – Window – Pubic subnet
   1. Windows
   2. Create new Key pair
   3. Our VPC (EU)
   4. Public subnet
   5. Auto-assign public IP - Yes
   6. Security Group Inbound rules:
      1. RDP - TCP – 3389 – Anywhere-IPv4
      2. HTTP - HTTP – 80– Anywhere-IPv4
      3. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
   7. Launch
8. Connect to instance
   1. Download RDP file
   2. Get password from .pem file (of key pair)
   3. Username: Administrator
   4. Browser check
9. Create NAT Gateway (NGW) – Name: EU-NGW
   1. Select public subnet
   2. Connectivity type: Public
   3. Attach Elastic IP
10. Create a Route Table for private subnet
    1. Associate to VPC (EU)
    2. Associate private subnet
    3. Add routes: 0.0.0.0/0 and select our EU-NGW.
11. Create an EC2 instance – Window – Private subnet
    1. Windows
    2. Use same Key pair
    3. Our VPC (EU)
    4. Private subnet
    5. Auto-assign public IP - No
    6. Security Group Inbound rules: Use same SG created with public instance
       1. RDP - TCP – 3389 – Anywhere-IPv4
       2. HTTP - HTTP – 80– Anywhere-IPv4
       3. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    7. Launch
12. Connect to instance using private IP
    1. Download RDP file
    2. Get password from .pem file (of key pair)
    3. Username: Administrator
    4. Browser check

## Configure VPC Peering

US – Requestor

EU – Acceptor

1. Switch to US-East N. Virginia
2. VPC Dashboard -> VPC Peering – Create New
3. Name: US-to-EU-Peering
4. VPC ID (Requestor): US VPC
5. VPC to peer with:
   * My Account
   * Another region
     + London
     + Select the EU VPC id (*copy the id and paste here*)
6. Create Peering Connection
7. Shows “pending acceptance”
8. Switch to London region
9. VPC Dashboard -> VPC Peering
10. Select connection (*pending acceptance*)
11. Actions -> Accept
12. Changes to Provisioning and then done in a few seconds
13. Switch to US region and refresh
14. Both networks are now successfully “peered”
15. Go to private instance of US VPC and try to ping private IP of private instance of EU VPC
16. Does not work. Requires Route Table entry on both sides
17. Switch to US region
    * VPC – Route Tables – Private Route Table – Routes – Edit routes
    * Add route:
      + Destination: 192.168.0.0/16 (*VPC path of EU*)
      + Target: Peering Connection – Select the peering connection
    * Go to private instance of US VPC and try to ping private IP of private instance of EU VPC
      + Does not work because path on the other side is not created
18. Switch to London region
    * VPC – Route Tables – Private Route Table – Routes – Edit routes
    * Add route:
      + Destination: 10.0.0.0/16 (*VPC path of US*)
      + Target: Peering Connection – Select the peering connection
    * Go to private instance of US VPC and try to ping private IP of private instance of EU VPC – still does not work
      + Windows firewall disables ICMP ping
      + Have to temporarily disable the Windows Firewall
      + On US private server, Windows Defender Firewall –> Turn Windows Defender Firewall on or off -> Turn off (*both options*) -> OK
      + Do this on both sides (*US and EU private instances*)
    * Go to private instance of US VPC and try to ping private IP of private instance of EU VPC – Works!
    * Go to private instance of EU VPC and try to ping private IP of private instance of US VPC – Works!

# Amazon VPC Quotas

<https://docs.aws.amazon.com/vpc/latest/userguide/amazon-vpc-limits.html>

The following tables list the quotas, formerly referred to as limits, for Amazon VPC resources per Region for your AWS account. Unless indicated otherwise, you can request an increase for these quotas. For some of these quotas, you can view your current quota using the **Limits** page of the Amazon EC2 console.

If you request a quota increase that applies per resource, we increase the quota for all resources in the Region.

## VPC and subnets

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| VPCs per Region | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-F678F1CE) | Increasing this quota increases the quota on internet gateways per Region by the same amount.  You can increase this limit so that you can have hundreds of VPCs per Region. |
| Subnets per VPC | 200 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-407747CB) |  |
| IPv4 CIDR blocks per VPC | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-83CA0A9D)  (up to 50) | This primary CIDR block and all secondary CIDR blocks count toward this quota. |
| IPv6 CIDR blocks per VPC | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-085A6257)  (up to 50) | The number of /56 CIDRs you can allocate to a single VPC. |

## DNS

Each EC2 instance can send 1024 packets per second per network interface to Route 53 Resolver (specifically the .2 address, such as 10.0.0.2 and 169.254.169.253). This quota cannot be increased. The number of DNS queries per second supported by Route 53 Resolver varies by the type of query, the size of the response, and the protocol in use. For more information and recommendations for a scalable DNS architecture, see the [AWS Hybrid DNS with Active Directory](https://d1.awsstatic.com/whitepapers/aws-hybrid-dns-with-active-directory.pdf) Technical Guide.

## Elastic IP addresses

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Elastic IP addresses per Region | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/ec2/quotas/L-0263D0A3) | This quota applies to individual AWS account VPCs and shared VPCs. |
| Elastic IP addresses per public NAT gateway | 2 | Yes. To request a quota increase up to 8, contact the AWS Support Center as described in [AWS service quotas](https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html) in the AWS General Reference. |  |

## Gateways

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Egress-only internet gateways per Region | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-45FE3B85) | To increase this quota, increase the quota for VPCs per Region.  You can attach only one egress-only internet gateway to a VPC at a time. |
| Internet gateways per Region | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-A4707A72) | To increase this quota, increase the quota for VPCs per Region.  You can attach only one internet gateway to a VPC at a time. |
| NAT gateways per Availability Zone | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-FE5A380F) | NAT gateways only count toward your quota in the pending, active, and deleting states. |
| Carrier gateways per VPC | 1 | No |  |

## Network ACLs

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Network ACLs per VPC | 200 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-B4A6D682) | You can associate one network ACL to one or more subnets in a VPC. |
| Rules per network ACL | 20 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-2AEEBF1A) | This is a one-way quota. This quota is enforced separately for IPv4 rules and IPv6 rules. Therefore, for an account with the default quota of 20 rules, a network ACL can have 20 inbound rules for IPv4 traffic and 20 inbound rules for IPv6 traffic. This quota includes the default deny rules (rule number 32767 for IPv4 and 32768 for IPv6, or an asterisk \* in the Amazon VPC console).  This quota can be increased up to a maximum of 40. However, network performance might be impacted due to the increased workload to process the additional rules. |

## Network interfaces

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Network interfaces per instance | Varies by instance type | No | For more information, see [Network interfaces per instance type](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html#AvailableIpPerENI). |
| Network interfaces per Region | 5,000 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-DF5E4CA3) | This quota applies to individual AWS account VPCs and shared VPCs. |

## Route tables

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Route tables per VPC | 200 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-589F43AA) | The main route table counts toward this quota. Note that if you request a quota increase for route tables, you may also want to request a quota increase for subnets. While route tables can be shared with multiple subnets, a subnet can only be associated with a single route table. |
| Routes per route table (non-propagated routes) | 50 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-93826ACB) | You can increase this quota up to a maximum of 1,000; however, network performance might be impacted. This quota is enforced separately for IPv4 routes and IPv6 routes.  If you have more than 125 routes, we recommend that you paginate calls to describe your route tables for better performance. |
| BGP advertised routes per route table (propagated routes) | 100 | No | If you require additional prefixes, advertise a default route. |

## Security groups

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| VPC security groups per Region | 2,500 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-E79EC296) | This quota applies to individual AWS account VPCs and shared VPCs.  If you increase this quota to more than 5,000 security groups in a Region, we recommend that you paginate calls to describe your security groups for better performance. |
| Inbound or outbound rules per security group | 60 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-0EA8095F) | This quota is enforced separately for IPv4 rules and IPv6 rules. Therefore, for an account with the default quota of 60 rules, a security group can have 60 inbound rules for IPv4 traffic and 60 inbound rules for IPv6 traffic. For more information, see [Security group size](https://docs.aws.amazon.com/vpc/latest/userguide/security-group-rules.html#security-group-size).  A quota change applies to both inbound and outbound rules. This quota multiplied by the quota for security groups per network interface cannot exceed 1,000. |
| Security groups per network interface | 5 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-2AFB9258)  (up to 16) | This quota multiplied by the quota for rules per security group cannot exceed 1,000. |

## VPC peering connections

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Active VPC peering connections per VPC | 50 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-7E9ECCDB)  (up to 125) | If you increase this quota, you should increase the number of entries per route table accordingly. |
| Outstanding VPC peering connection requests | 25 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-DC9F7029) | This is the number of outstanding VPC peering connection requests made from your account. |
| Expiry time for an unaccepted VPC peering connection request | 1 week (168 hours) | No |  |

For more information, see [VPC peering limitations](https://docs.aws.amazon.com/vpc/latest/peering/vpc-peering-basics.html#vpc-peering-limitations) in the Amazon VPC Peering Guide.

## VPC endpoints

| **Name** | **Default** | **Adjustable** | **Comments** |
| --- | --- | --- | --- |
| Gateway VPC endpoints per Region | 20 | [Yes](https://console.aws.amazon.com/servicequotas/home/services/vpc/quotas/L-1B52E74A) | You can't have more than 255 gateway endpoints per VPC. |
| Interface and Gateway Load Balancer endpoints per VPC | 50 | Yes | This is the combined quota for the maximum number of interface endpoints and Gateway Load Balancer endpoints in a VPC. To increase this quota, contact AWS Support. |
| VPC endpoint policy size | 20,480 characters | No | This quota includes white space. |

# Network Access Control List (NACL)

NACL refers to Network Access Control List, which helps provide a layer of security to the Amazon Web Services stack.

NACL helps in providing a firewall thereby helping secure the VPCs and subnets. It helps provide a security layer which controls and efficiently manages the traffic that moves around in the subnets. It is an optional layer for VPC, which adds another security layer to the Amazon service.

VPC refers to Virtual private Cloud, which can be visualized as a container that stores subnets. Subnets can be considered as a container, which helps store data.

## **The Stateless Beauty of AWS NACLs**

Before exploring the best practices of AWS NACLs, it is important to understand its basic characteristics as well as the ability to fine-tune traffic through its stateless behavior.

Unlike SGs that are stateful, AWS NACLs are stateless. On that account, changes applicable to an incoming rule will not be applicable to the outgoing rule. That is, if you want your instances to communicate over port 80 (HTTP), then you have to add an inbound as well as an outbound rule allowing port 80.

## Control traffic to subnets using Network ACLs

<https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html>

A network access control list (ACL) allows or denies specific inbound or outbound traffic at the subnet level. You can use the default network ACL for your VPC, or you can create a custom network ACL for your VPC with rules that are similar to the rules for your security groups in order to add an additional layer of security to your VPC.

There is no additional charge for using network ACLs.

### Network ACL basics

The following are the basic things that you need to know about network ACLs:

* Your VPC automatically comes with a modifiable default network ACL. By default, it allows all inbound and outbound IPv4 traffic and, if applicable, IPv6 traffic.
* You can create a custom network ACL and associate it with a subnet to allow or deny specific inbound or outbound traffic at the subnet level.
* Each subnet in your VPC must be associated with a network ACL. If you don't explicitly associate a subnet with a network ACL, the subnet is automatically associated with the default network ACL.
* You can associate a network ACL with multiple subnets. However, a subnet can be associated with only one network ACL at a time. When you associate a network ACL with a subnet, the previous association is removed.
* A network ACL has inbound rules and outbound rules. Each rule can either allow or deny traffic. Each rule has a number from 1 to 32766. We evaluate the rules in order, starting with the lowest numbered rule, when deciding whether allow or deny traffic. If the traffic matches a rule, the rule is applied and we do not evaluate any additional rules. We recommend that you start by creating rules in increments (for example, increments of 10 or 100) so that you can insert new rules later on, if needed.
* We evaluate the network ACL rules when traffic enters and leaves the subnet, not as it is routed within a subnet.
* Network ACLs are stateless, which means that responses to allowed inbound traffic are subject to the rules for outbound traffic (and vice versa).
* Network ACLs can't block DNS requests to or from the Route 53 Resolver (also known as the VPC+2 IP address or AmazonProvidedDNS). To filter DNS requests through the Route 53 Resolver, you can enable [Route 53 Resolver DNS Firewall](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resolver-dns-firewall.html) in the Amazon Route 53 Developer Guide.
* Network ACLs can't block traffic to the Instance Metadata Service (IMDS). To manage access to IMDS, see [Configure the instance metadata options](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-options.html) in the Amazon EC2 User Guide.

There are quotas (also known as limits) for the number of network ACLs per VPC and the number of rules per network ACL. For more information, see [Amazon VPC quotas](https://docs.aws.amazon.com/vpc/latest/userguide/amazon-vpc-limits.html).

### Network ACL rules

You can add or remove rules from the default network ACL, or create additional network ACLs for your VPC. When you add or remove rules from a network ACL, the changes are automatically applied to the subnets that it's associated with.

The following are the parts of a network ACL rule:

* **Rule number**. Rules are evaluated starting with the lowest numbered rule. As soon as a rule matches traffic, it's applied regardless of any higher-numbered rule that might contradict it.
* **Type**. The type of traffic; for example, SSH. You can also specify all traffic or a custom range.
* **Protocol**. You can specify any protocol that has a standard protocol number. For more information, see [Protocol Numbers](http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml). If you specify ICMP as the protocol, you can specify any or all of the ICMP types and codes.
* **Port range**. The listening port or port range for the traffic. For example, 80 for HTTP traffic.
* **Source**. [Inbound rules only] The source of the traffic (CIDR range).
* **Destination**. [Outbound rules only] The destination for the traffic (CIDR range).
* **Allow/Deny**. Whether to allow or deny the specified traffic.

If you add a rule using a command line tool or the Amazon EC2 API, the CIDR range is automatically modified to its canonical form. For example, if you specify 100.68.0.18/18 for the CIDR range, we create a rule with a 100.68.0.0/18 CIDR range.

### Default network ACL

The default network ACL is configured to allow all traffic to flow in and out of the subnets with which it is associated. Each network ACL also includes a rule whose rule number is an asterisk. This rule ensures that if a packet doesn't match any of the other numbered rules, it's denied. You can't modify or remove this rule.

The following is an example default network ACL for a VPC that supports IPv4 only.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Inbound** | | | | | |
| **Rule #** | **Type** | **Protocol** | **Port range** | **Source** | **Allow/Deny** |
| 100 | All IPv4 traffic | All | All | 0.0.0.0/0 | ALLOW |
| \* | All IPv4 traffic | All | All | 0.0.0.0/0 | DENY |
| **Outbound** | | | | | |
| **Rule #** | **Type** | **Protocol** | **Port range** | **Destination** | **Allow/Deny** |
| 100 | All IPv4 traffic | All | All | 0.0.0.0/0 | ALLOW |
| \* | All IPv4 traffic | All | All | 0.0.0.0/0 | DENY |

## Work with network ACLs

### Determine network ACL associations

You can use the Amazon VPC console to determine the network ACL that's associated with a subnet. Network ACLs can be associated with more than one subnet, so you can also determine which subnets are associated with a network ACL.

To determine which network ACL is associated with a subnet

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Subnets**, and then select the subnet.

The network ACL associated with the subnet is included in the **Network ACL** tab, along with the network ACL's rules.

To determine which subnets are associated with a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**. The **Associated With** column indicates the number of associated subnets for each network ACL.
3. Select a network ACL.
4. In the details pane, choose **Subnet Associations** to display the subnets that are associated with the network ACL.

### Create a network ACL

You can create a custom network ACL for your VPC. By default, a network ACL that you create blocks all inbound and outbound traffic until you add rules, and is not associated with a subnet until you explicitly associate it with one.

To create a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**.
3. Choose **Create Network ACL**.
4. In the **Create Network ACL** dialog box, optionally name your network ACL, and select the ID of your VPC from the **VPC** list. Then choose **Yes, Create**.

### Add and delete rules

When you add or delete a rule from an ACL, any subnets that are associated with the ACL are subject to the change. You don't have to terminate and relaunch the instances in the subnet. The changes take effect after a short period.

If you're using the Amazon EC2 API or a command line tool, you can't modify rules. You can only add and delete rules. If you're using the Amazon VPC console, you can modify the entries for existing rules. The console removes the existing rule and adds a new rule for you. If you need to change the order of a rule in the ACL, you must add a new rule with the new rule number, and then delete the original rule.

To add rules to a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**.
3. In the details pane, choose either the **Inbound Rules** or **Outbound Rules** tab, depending on the type of rule that you need to add, and then choose **Edit**.
4. In **Rule #**, enter a rule number (for example, 100). The rule number must not already be in use in the network ACL. We process the rules in order, starting with the lowest number.

We recommend that you leave gaps between the rule numbers (such as 100, 200, 300), rather than using sequential numbers (101, 102, 103). This makes it easier add a new rule without having to renumber the existing rules.

1. Select a rule from the **Type** list. For example, to add a rule for HTTP, choose **HTTP**. To add a rule to allow all TCP traffic, choose **All TCP**. For some of these options (for example, HTTP), we fill in the port for you. To use a protocol that's not listed, choose **Custom Protocol Rule**.
2. (Optional) If you're creating a custom protocol rule, select the protocol's number and name from the **Protocol** list. For more information, see [IANA List of Protocol Numbers](http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml).
3. (Optional) If the protocol you selected requires a port number, enter the port number or port range separated by a hyphen (for example, 49152-65535).
4. In the **Source** or **Destination** field (depending on whether this is an inbound or outbound rule), enter the CIDR range that the rule applies to.
5. From the **Allow/Deny** list, select **ALLOW** to allow the specified traffic or **DENY** to deny the specified traffic.
6. (Optional) To add another rule, choose **Add another rule**, and repeat steps 4 to 9 as required.
7. When you are done, choose **Save**.

To delete a rule from a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**, and then select the network ACL.
3. In the details pane, select either the **Inbound Rules** or **Outbound Rules** tab, and then choose **Edit**. Choose **Remove** for the rule you want to delete, and then choose **Save**.

### Associate a subnet with a network ACL

To apply the rules of a network ACL to a particular subnet, you must associate the subnet with the network ACL. You can associate a network ACL with multiple subnets. However, a subnet can be associated with only one network ACL. Any subnet that is not associated with a particular ACL is associated with the default network ACL by default.

To associate a subnet with a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**, and then select the network ACL.
3. In the details pane, on the **Subnet Associations** tab, choose **Edit**. Select the **Associate** check box for the subnet to associate with the network ACL, and then choose **Save**.

### Disassociate a network ACL from a subnet

You can disassociate a custom network ACL from a subnet. When the subnet has been disassociated from the custom network ACL, it is then automatically associated with the default network ACL.

To disassociate a subnet from a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**, and then select the network ACL.
3. In the details pane, choose the **Subnet Associations** tab.
4. Choose **Edit**, and then deselect the **Associate** check box for the subnet. Choose **Save**.

### Change a subnet's network ACL

You can change the network ACL that's associated with a subnet. For example, when you create a subnet, it is initially associated with the default network ACL. You might want to instead associate it with a custom network ACL that you've created.

After changing a subnet's network ACL, you don't have to terminate and relaunch the instances in the subnet. The changes take effect after a short period.

To change a subnet's network ACL association

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Subnets**, and then select the subnet.
3. Choose the **Network ACL** tab, and then choose **Edit**.
4. From the **Change to** list, select the network ACL to associate the subnet with, and then choose **Save**.

### Delete a network ACL

You can delete a network ACL only if there are no subnets associated with it. You can't delete the default network ACL.

To delete a network ACL

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Network ACLs**.
3. Select the network ACL, and then choose **Delete**.
4. In the confirmation dialog box, choose **Yes, Delete**.

# Create a MySQL RDS

## Create the DB

1. Login to the console http://console.aws.amazon.com/rds
2. Select create database
3. Standard create
4. MySQL
5. Templates -> Free Tier
6. DB instance identifier (name)
7. Master username
8. Master password
9. Confirm password
10. DB instance class -> Burstable classes -> db.t2.micro
11. General purpose SSD (gp2)
12. Enable storage autoscaling: No
13. Max threshold: 1000
14. Don’t connect to EC2 instance
15. IPv4
16. Default VPC
17. Public access: Yes
18. Create New Security group
19. AZ: no preference
20. Additional config -> DB port: 3306
21. DB auth: password auth
22. Additional config -> db name
23. Automated backups?
24. Backup Retention period.
25. Estimated cost
26. Create database

## Connect to the DB

1. Open MySQL Workbench on local machine
2. Database -> Connect to DB
3. Enter the db endpoint and port
4. Master username
5. Click on “Store in vault” for password and enter password
6. OK
7. Show schemas

# VPC with EC2 Instance and RDS

## Create a VPC for use with a DB instance (dual-stack mode)

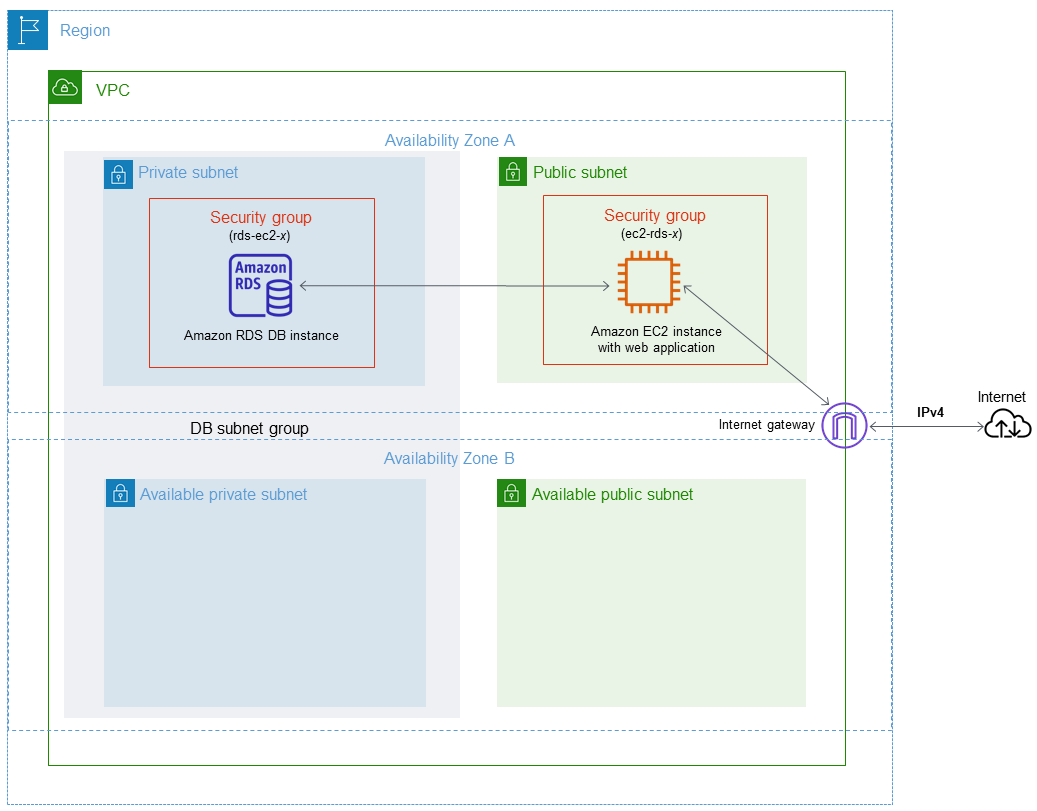
<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateVPC.html>

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT_WebAppWithRDS.html>

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateDBInstance.html>

A common scenario includes a DB instance in a virtual private cloud (VPC) based on the Amazon VPC service. This VPC shares data with a web server that is running in the same VPC. In this tutorial, you create the VPC for this scenario.

The following diagram shows this scenario.



Your DB instance needs to be available only to your web server, and not to the public internet. Thus, you create a VPC with both public and private subnets. The web server is hosted in the public subnet, so that it can reach the public internet. The DB instance is hosted in a private subnet. The web server can connect to the DB instance because it is hosted within the same VPC. But the DB instance isn't available to the public internet, providing greater security.

This tutorial configures an additional public and private subnet in a separate Availability Zone. These subnets aren't used by the tutorial. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easier to switch to a Multi-AZ DB instance deployment in the future.

### Create a VPC with private and public subnets

Use the following procedure to create a VPC with both public and private subnets.

To create a VPC and subnets

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the top-right corner of the AWS Management Console, choose the Region to create your VPC in. This example uses the US West (Oregon) Region.
3. In the upper-left corner, choose **VPC dashboard**. To begin creating a VPC, choose **Create VPC**.
4. For **Resources to create** under **VPC settings**, choose **VPC and more**.
5. For the **VPC settings**, set these values:

* Name tag auto-generation – **tutorial**
* IPv4 CIDR block – **10.0.0.0/16**
* IPv6 CIDR block – No IPv6 CIDR block
* Tenancy – Default
* Number of Availability Zones (AZs) – 2
* Customize AZs – Keep the default values.
* Number of public subnet – 2
* Number of private subnets – 2
* Customize subnets CIDR blocks – Keep the default values.
* NAT gateways ($) – None
* VPC endpoints – None
* DNS options – Keep the default values.

**Note**: Amazon RDS requires at least two subnets in two different Availability Zones to support Multi-AZ DB instance deployments. This tutorial creates a Single-AZ deployment, but the requirement makes it easier to convert to a Multi-AZ DB instance deployment in the future.

1. Choose Create VPC.

### Create a VPC security group for a public web server

Next, you create a security group for public access. To connect to public EC2 instances in your VPC, you add inbound rules to your VPC security group. These allow traffic to connect from the internet.

To create a VPC security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
   * Security group name: **tutorial-securitygroup**
   * Description: **Tutorial Security Group**
   * VPC: Choose the VPC that you created earlier, for example: vpc-***identifier*** (tutorial-vpc)
4. Add inbound rules to the security group.
   1. Determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at [https://checkip.amazonaws.com](https://checkip.amazonaws.com/). An example of an IP address is 203.0.113.25/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, find the range of IP addresses used by client computers.

**Warning**: If you use 0.0.0.0/0 for SSH access, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instances using SSH.

* 1. In the **Inbound rules** section, choose **Add rule**.
  2. Set the following values for your new inbound rule to allow SSH access to your Amazon EC2 instance. If you do this, you can connect to your Amazon EC2 instance to install the web server and other utilities. You also connect to your EC2 instance to upload content for your web server.
     + **Type:** **SSH**
     + **Source:** The IP address or range from Step a, for example: **203.0.113.25/32**.
  3. Choose **Add rule**.
  4. Set the following values for your new inbound rule to allow HTTP access to your web server:
     + **Type:** **HTTP**
     + **Source:** **0.0.0.0/0**

1. Choose **Create security group** to create the security group.

Note the security group ID because you need it later in this tutorial.

### Create a VPC security group for a private DB instance

To keep your DB instance private, create a second security group for private access. To connect to private DB instancesin your VPC, you add inbound rules to your VPC security group that allow traffic from your web server only.

To create a VPC security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
   * **Security group name:** **tutorial-db-securitygroup**
   * **Description:** **Tutorial DB Instance Security Group**
   * **VPC:** Choose the VPC that you created earlier, for example: **vpc-*identifier* (tutorial-vpc)**
4. Add inbound rules to the security group.
   1. In the **Inbound rules** section, choose **Add rule**.
   2. Set the following values for your new inbound rule to allow MySQL traffic on port 3306 from your Amazon EC2 instance. If you do this, you can connect from your web server to your DB instance. By doing so, you can store and retrieve data from your web application to your database.
      * **Type:** **MySQL/Aurora**
      * **Source:** The identifier of the **tutorial-securitygroup** security group that you created previously in this tutorial, for example: **sg-9edd5cfb**.
5. Choose **Create security group** to create the security group.

### Create a DB subnet group

A DB subnet group is a collection of subnets that you create in a VPC and that you then designate for your DB instances. A DB subnet group makes it possible for you to specify a particular VPC when creating DB instances.

To create a DB subnet group

1. Identify the private subnets for your database in the VPC.
   1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
   2. Choose **VPC Dashboard**, and then choose **Subnets**.
   3. Note the subnet IDs of the subnets named **tutorial-subnet-private1-us-west-2a** and **tutorial-subnet-private2-us-west-2b**.

You need the subnet IDs when you create your DB subnet group.

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

Make sure that you connect to the Amazon RDS console, not to the Amazon VPC console.

1. In the navigation pane, choose **Subnet groups**.
2. Choose **Create DB subnet group**.
3. On the **Create DB subnet group** page, set these values in **Subnet group details**:

* Name: **tutorial-db-subnet-group**
* Description: **Tutorial DB Subnet Group**
* VPC: tutorial-vpc (vpc-***identifier***)

1. In the **Add subnets** section, choose the **Availability Zones and Subnets**.
2. For this tutorial, choose **us-west-2a** and **us-west-2b** (or whichever you selected earlier) for the **Availability Zones**. For **Subnets**, choose the private subnets you identified in the previous step.
3. Choose **Create**.

Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can choose the DB subnet group to see details in the details pane at the bottom of the window. These details include all of the subnets associated with the group.

## Create a web server and an Amazon RDS DB instance

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT_WebAppWithRDS.html>

This tutorial helps you install an Apache web server with PHP and create a MySQL database. The web server runs on an Amazon EC2 instance using Amazon Linux, and the MySQL database is a MySQL DB instance. Both the Amazon EC2 instance and the DB instance run in a virtual private cloud (VPC) based on the Amazon VPC service.

**Note**: This tutorial works with Amazon Linux and might not work for other versions of Linux such as Ubuntu.

### Launch an EC2 instance

Create an Amazon EC2 instance in the public subnet of your VPC.

To launch an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region where you want to create the EC2 instance. It should be the same as the one where you created your VPC in the previous step.
3. Choose **EC2 Dashboard**, and then choose **Launch instance.**
4. Choose the following settings in the **Launch an instance** page.
   1. Under Name and tags, for Name, enter **ec2-database-connect**.
   2. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2 AMI**. Keep the defaults for the other choices.
   3. Under **Instance type**, choose **t2.micro**.
   4. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair**window to create it.

For more information about creating a new key pair, see [Create a key pair](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html#create-a-key-pair) in the Amazon EC2 User Guide for Linux Instances.

* 1. Under **Network settings**, set these values and keep the other values as their defaults:
* For VPC, select the VPC and public subnet you created in the section [Create a VPC for use with DB instance (dual-stack mode).](#_Create_a_VPC)
* For **Allow SSH traffic from**, choose the source of SSH connections to the EC2 instance.

You can choose **My IP** if the displayed IP address is correct for SSH connections.

Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at [https://checkip.amazonaws.com](https://checkip.amazonaws.com/). An example of an IP address is 203.0.113.25/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

**Warning:** If you use 0.0.0.0/0 for SSH access, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instances using SSH.

* Turn on Allow HTTPs traffic from the internet.
* Turn on Allow HTTP traffic from the internet.
  1. Leave the default values for the remaining sections.
  2. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

1. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: i-03a6ad47e97ba9dc5.
2. Choose **View all instances** to find your instance.
3. Wait until **Instance state** for your instance is **Running** before continuing.
4. Complete [Create a DB instance](#_Create_a_DB).

### Create a DB instance

Create an Amazon RDS for MySQL DB instance that maintains the data used by a web application.

To create a MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, check the AWS Region. It should be the same as the one where you created your EC2 instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, choose **Standard create**.
6. For **Engine type**, choose **MySQL**.
7. For **Templates**, choose **Free tier**.
8. In the **Availability and durability** section, keep the defaults.
9. In the **Settings** section, set these values:

* DB instance identifier – Type **tutorial-db-instance**.
* Master username – Type **tutorial\_user**.
* Auto generate a password – Leave the option turned off.
* Master password – Type a password.
* Confirm password – Retype the password.

1. In the **Instance configuration** section, set these values:

* Burstable classes (includes t classes)
* db.t3.micro

1. In the **Storage** section, keep the defaults.
2. In the **Connectivity** section, set these values and keep the other values as their defaults:

* For Compute resource, choose Connect to an EC2 compute resource.
* For EC2 instance, choose the EC2 instance you [created previously](#_Launch_an_EC2), such as tutorial-ec2-instance-web-server.

1. In the Database authentication section, make sure Password authentication is selected.
2. Open the Additional configuration section, and enter **sample** for Initial database name. Keep the default settings for the other options.
3. To create your MySQL DB instance, choose Create database.

Your new DB instance appears in the **Databases** list with the status **Creating**.

1. Wait for the **Status** of your new DB instance to show as **Available**. Then choose the DB instance name to show its details.
2. In the **Connectivity & security** section, view the **Endpoint** and **Port** of the DB instance.

Note the endpoint and port for your DB instance. You use this information to connect your web server to your DB instance. For e.g.; tutorial-db-instance.crkl4etghhxm.ap-south-1.rds.amazonaws.com:3306

1. Complete [Install a web server on your EC2 instance](#_Install_a_web).

### Install a web server on your EC2 instance

Install a web server on the EC2 instance you created in [Launch an EC2 instance](#_Launch_an_EC2). The web server connects to the Amazon RDS DB instance that you created in [Create a DB instance](#_Create_a_DB).

#### Install an Apache web server with PHP and MariaDB

To connect to your EC2 instance and install the Apache web server with PHP

1. Connect to the EC2 instance that you created earlier by following the steps in [Connect to your Linux instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html).
2. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do this, use the following command.

**Note**: The -y option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

sudo yum update -y

1. After the updates complete, install the PHP software using the amazon-linux-extras install command. This command installs multiple software packages and related dependencies at the same time.

sudo amazon-linux-extras install php8.0 mariadb10.5

If you receive an error stating sudo: amazon-linux-extras: command not found, your instance wasn't launched with an Amazon Linux 2 AMI. You might be using the Amazon Linux AMI instead. You can view your version of Amazon Linux using the following command.

cat /etc/system-release

1. Install the Apache web server.

sudo yum install -y httpd

1. Start the web server with the command shown following.

sudo systemctl start httpd

You can test that your web server is properly installed and started. To do this, enter the public Domain Name System (DNS) name of your EC2 instance in the address bar of a web browser, for example: http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com. If your web server is running, then you see the Apache test page.

If you don't see the Apache test page, check your inbound rules for the VPC security group that you created in [Tutorial: Create a VPC for use with a DB instance (IPv4 only)](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateVPC.html). Make sure that your inbound rules include one allowing HTTP (port 80) access for the IP address to connect to the web server.

**Note** : The Apache test page appears only when there is no content in the document root directory, /var/www/html. After you add content to the document root directory, your content appears at the public DNS address of your EC2 instance. Before this point, it appears on the Apache test page.

1. Configure the web server to start with each system boot using the systemctl command.

sudo systemctl enable httpd

To allow ec2-user to manage files in the default root directory for your Apache web server, modify the ownership and permissions of the /var/www directory. There are many ways to accomplish this task. In this tutorial, you add ec2-user to the apache group, to give the apache group ownership of the /var/www directory and assign write permissions to the group.

To set file permissions for the Apache web server

1. Add the ec2-user user to the apache group.

sudo usermod -a -G apache ec2-user

1. Log out to refresh your permissions and include the new apache group.

exit

1. Log back in again and verify that the apache group exists with the groups command.

groups

Your output looks similar to the following:

ec2-user adm wheel apache systemd-journal

1. Change the group ownership of the /var/www directory and its contents to the apache group.

sudo chown -R ec2-user:apache /var/www

1. Change the directory permissions of /var/www and its subdirectories to add group write permissions and set the group ID on subdirectories created in the future.

sudo chmod 2775 /var/www

find /var/www -type d -exec sudo chmod 2775 {} \;

1. Recursively change the permissions for files in the /var/www directory and its subdirectories to add group write permissions.

find /var/www -type f -exec sudo chmod 0664 {} \;

Now, ec2-user (and any future members of the apache group) can add, delete, and edit files in the Apache document root. This makes it possible for you to add content, such as a static website or a PHP application.

**Note**: A web server running the HTTP protocol provides no transport security for the data that it sends or receives. When you connect to an HTTP server using a web browser, much information is visible to eavesdroppers anywhere along the network pathway. This information includes the URLs that you visit, the content of web pages that you receive, and the contents (including passwords) of any HTML forms.

The best practice for securing your web server is to install support for HTTPS (HTTP Secure). This protocol protects your data with SSL/TLS encryption. For more information, see [Tutorial: Configure SSL/TLS with the Amazon Linux AMI](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-ami.html) in the Amazon EC2 User Guide.

#### Connect your Apache web server to your DB instance

Next, you add content to your Apache web server that connects to your Amazon RDS DB instance.

To add content to the Apache web server that connects to your DB instance

1. While still connected to your EC2 instance, change the directory to /var/www and create a new subdirectory named inc.

cd /var/www

mkdir inc

cd inc

1. Create a new file in the inc directory named dbinfo.inc, and then edit the file by calling nano (or the editor of your choice).

>dbinfo.inc

nano dbinfo.inc

1. Add the following contents to the dbinfo.inc file. Here, *db\_instance\_endpoint:port* is your DB instance endpoint, without the port, and *master password* is the master password for your DB instance.

**Note**: We recommend placing the user name and password information in a folder that isn't part of the document root for your web server. Doing this reduces the possibility of your security information being exposed.

<?php

define('DB\_SERVER', '*db\_instance\_endpoint:port*');

define('DB\_USERNAME', 'tutorial\_user');

define('DB\_PASSWORD', '*master password*');

define('DB\_DATABASE', 'sample');

?>

1. Save and close the dbinfo.inc file. If you are using nano, save and close the file by using Ctrl+S and Ctrl+X.
2. Change the directory to /var/www/html.

cd /var/www/html

1. Create a new file in the html directory named SamplePage.php, and then edit the file by calling nano (or the editor of your choice).

>SamplePage.php

nano SamplePage.php

1. Add the following contents to the SamplePage.php file:

<?php include "../inc/dbinfo.inc"; ?>

<html>

<body>

<h1>Sample page</h1>

<?php

*/\* Connect to MySQL and select the database. \*/*

$connection = mysqli\_connect(DB\_SERVER, DB\_USERNAME, DB\_PASSWORD);

if (mysqli\_connect\_errno()) echo "Failed to connect to MySQL: " . mysqli\_connect\_error();

$database = mysqli\_select\_db($connection, DB\_DATABASE);

*/\* Ensure that the EMPLOYEES table exists. \*/*

VerifyEmployeesTable($connection, DB\_DATABASE);

*/\* If input fields are populated, add a row to the EMPLOYEES table. \*/*

$employee\_name = htmlentities($\_POST['NAME']);

$employee\_address = htmlentities($\_POST['ADDRESS']);

if (strlen($employee\_name) || strlen($employee\_address)) {

AddEmployee($connection, $employee\_name, $employee\_address);

}

?>

<!-- Input form -->

<form action="<?PHP echo $\_SERVER['SCRIPT\_NAME'] ?>" method="POST">

<table border="0">

<tr>

<td>NAME</td>

<td>ADDRESS</td>

</tr>

<tr>

<td>

<input type="text" name="NAME" maxlength="45" size="30" />

</td>

<td>

<input type="text" name="ADDRESS" maxlength="90" size="60" />

</td>

<td>

<input type="submit" value="Add Data" />

</td>

</tr>

</table>

</form>

<!-- Display table data. -->

<table border="1" cellpadding="2" cellspacing="2">

<tr>

<td>ID</td>

<td>NAME</td>

<td>ADDRESS</td>

</tr>

<?php

$result = mysqli\_query($connection, "SELECT \* FROM EMPLOYEES");

while($query\_data = mysqli\_fetch\_row($result)) {

echo "<tr>";

echo "<td>",$query\_data[0], "</td>",

"<td>",$query\_data[1], "</td>",

"<td>",$query\_data[2], "</td>";

echo "</tr>";

}

?>

</table>

<!-- Clean up. -->

<?php

mysqli\_free\_result($result);

mysqli\_close($connection);

?>

</body>

</html>

<?php

*/\* Add an employee to the table. \*/*

function AddEmployee($connection, $name, $address) {

$n = mysqli\_real\_escape\_string($connection, $name);

$a = mysqli\_real\_escape\_string($connection, $address);

$query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

if(!mysqli\_query($connection, $query)) echo("<p>Error adding employee data.</p>");

}

*/\* Check whether the table exists and, if not, create it. \*/*

function VerifyEmployeesTable($connection, $dbName) {

if(!TableExists("EMPLOYEES", $connection, $dbName))

{

$query = "CREATE TABLE EMPLOYEES (

ID int(11) UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

NAME VARCHAR(45),

ADDRESS VARCHAR(90)

)";

if(!mysqli\_query($connection, $query)) echo("<p>Error creating table.</p>");

}

}

*/\* Check for the existence of a table. \*/*

function TableExists($tableName, $connection, $dbName) {

$t = mysqli\_real\_escape\_string($connection, $tableName);

$d = mysqli\_real\_escape\_string($connection, $dbName);

$checktable = mysqli\_query($connection,

"SELECT TABLE\_NAME FROM information\_schema.TABLES WHERE TABLE\_NAME = '$t' AND TABLE\_SCHEMA = '$d'");

if(mysqli\_num\_rows($checktable) > 0) return true;

return false;

}

?>

1. Save and close the SamplePage.php file.
2. Verify that your web server successfully connects to your DB instance by opening a web browser and browsing to http://*EC2 instance endpoint*/SamplePage.php, for example: http://ec2-55-122-41-31.us-west-2.compute.amazonaws.com/SamplePage.php.

You can use SamplePage.php to add data to your DB instance. The data that you add is then displayed on the page. To verify that the data was inserted into the table, install MySQL client on the Amazon EC2 instance. Then connect to the DB instance and query the table.

For information about installing the MySQL client and connecting to a DB instance, see [Connecting to a DB instance running the MySQL database engine](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html).

To make sure that your DB instance is as secure as possible, verify that sources outside of the VPC can't connect to your DB instance.

After you have finished testing your web server and your database, you should delete your DB instance and your Amazon EC2 instance.

* To delete a DB instance, follow the instructions in [Deleting a DB instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_DeleteInstance.html). You don't need to create a final snapshot.
* To terminate an Amazon EC2 instance, follow the instruction in [Terminate your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/terminating-instances.html) in the Amazon EC2 User Guide.

## Enabling Automated Backups for RDS

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithAutomatedBackups.html>

If your database doesn't have automated backups enabled, you can enable them at any time. You enable automated backups by setting the backup retention period to a positive nonzero value. When automated backups are turned on, your RDS instance and database is taken offline and a backup is immediately created.

To enable automated backups immediately

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB instance or Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**.
4. For **Backup retention period**, choose a positive nonzero value, for example 3 days.
5. Choose **Continue**.
6. Choose **Apply immediately**.
7. Choose **Modify DB instance** or **Modify cluster** to save your changes and enable automated backups.

### Viewing automated backups

To view your automated backups, choose **Automated backups** in the navigation pane. To view individual snapshots associated with an automated backup, choose **Snapshots** in the navigation pane. Alternatively, you can describe individual snapshots associated with an automated backup. From there, you can restore a DB instance directly from one of those snapshots.

### Retaining automated backups

**Note**: You can only retain automated backups of DB instances, not Multi-AZ DB clusters.

When you delete a DB instance, you can retain automated backups.

Retained automated backups contain system snapshots and transaction logs from a DB instance. They also include your DB instance properties like allocated storage and DB instance class, which are required to restore it to an active instance.

You can retain automated backups for RDS instances running the MySQL, MariaDB, PostgreSQL, Oracle, and Microsoft SQL Server engines.

You can restore or remove retained automated backups using the AWS Management Console, RDS API, and AWS CLI.

### Disabling automated backups

You might want to temporarily disable automated backups in certain situations, for example while loading large amounts of data.

**Important:** We highly discourage disabling automated backups because it disables point-in-time recovery. Disabling automatic backups for a DB instance or Multi-AZ DB cluster deletes all existing automated backups for the database. If you disable and then re-enable automated backups, you can restore starting only from the time you re-enabled automated backups.

To disable automated backups immediately

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB instance or Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**.
4. For **Backup retention period**, choose **0 days**.
5. Choose **Continue**.
6. Choose **Apply immediately**.
7. Choose **Modify DB instance** or **Modify cluster** to save your changes and disable automated backups.

## Enabling cross-Region automated backups

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReplicateBackups.html>

For added disaster recovery capability, you can configure your Amazon RDS database instance to replicate snapshots and transaction logs to a destination AWS Region of your choice. When backup replication is configured for a DB instance, RDS initiates a cross-Region copy of all snapshots and transaction logs as soon as they are ready on the DB instance.

DB snapshot copy charges apply to the data transfer. After the DB snapshot is copied, standard charges apply to storage in the destination Region.

You can enable backup replication on new or existing DB instances using the Amazon RDS console. You can also use the start-db-instance-automated-backups-replication AWS CLI command or the StartDBInstanceAutomatedBackupsReplication RDS API operation. You can replicate up to 20 backups to each destination AWS Region for each AWS account.

You can enable backup replication for a new or existing DB instance:

* For a new DB instance, enable it when you launch the instance. For more information, see [Settings for DB instances](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html#USER_CreateDBInstance.Settings).
* For an existing DB instance, use the following procedure.

To enable backup replication for an existing DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Automated backups**.
3. On the **Current Region** tab, choose the DB instance for which you want to enable backup replication.
4. For **Actions**, choose **Manage cross-Region replication**.
5. Under **Backup replication**, choose **Enable replication to another AWS Region**.
6. Choose the **Destination Region**.
7. Choose the **Replicated backup retention period**.
8. If you've enabled encryption on the source DB instance, choose the **AWS KMS key** for encrypting the backups.
9. Choose **Save**.

In the source Region, replicated backups are listed on the **Current Region** tab of the **Automated backups** page. In the destination Region, replicated backups are listed on the **Replicated backups** tab of the **Automated backups** page.

### Stopping automated backup replication

You can stop backup replication for DB instances using the Amazon RDS console. You can also use the stop-db-instance-automated-backups-replication AWS CLI command or the StopDBInstanceAutomatedBackupsReplication RDS API operation.

Replicated backups are retained, subject to the backup retention period set when they were created.

Stop backup replication from the **Automated backups** page in the source Region.

To stop backup replication to an AWS Region

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the source Region from the **Region selector**.
3. In the navigation pane, choose **Automated backups**.
4. On the **Current Region** tab, choose the DB instance for which you want to stop backup replication.
5. For **Actions**, choose **Manage cross-Region replication**.
6. Under **Backup replication**, clear the **Enable replication to another AWS Region** check box.
7. Choose **Save**.

Replicated backups are listed on the **Retained** tab of the **Automated backups** page in the destination Region.

### Deleting replicated backups

You can delete replicated backups for DB instances using the Amazon RDS console. You can also use the delete-db-instance-automated-backups AWS CLI command or the DeleteDBInstanceAutomatedBackup RDS API operation.

Delete replicated backups in the destination Region from the **Automated backups** page.

To delete replicated backups

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the destination Region from the **Region selector**.
3. In the navigation pane, choose **Automated backups**.
4. On the **Replicated backups** tab, choose the DB instance for which you want to delete the replicated backups.
5. For **Actions**, choose **Delete**.
6. On the confirmation page, enter **delete me** and choose **Delete**.

## Creating a DB snapshot

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html>

Amazon RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. Creating this DB snapshot on a Single-AZ DB instance results in a brief I/O suspension that can last from a few seconds to a few minutes, depending on the size and class of your DB instance. For MariaDB, MySQL, Oracle, and PostgreSQL, I/O activity is not suspended on your primary during backup for Multi-AZ deployments, because the backup is taken from the standby. For SQL Server, I/O activity is suspended briefly during backup for Multi-AZ deployments.

When you create a DB snapshot, you need to identify which DB instance you are going to back up, and then give your DB snapshot a name so you can restore from it later. The amount of time it takes to create a snapshot varies with the size of your databases. Since the snapshot includes the entire storage volume, the size of files, such as temporary files, also affects the amount of time it takes to create the snapshot.

**Note**: Your DB instance must be in the available state to take a DB snapshot.

For PostgreSQL DB instances, data in unlogged tables might not be restored from snapshots. For more information, see [Best practices for working with PostgreSQL](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_BestPractices.html#CHAP_BestPractices.PostgreSQL).

Unlike automated backups, manual snapshots aren't subject to the backup retention period. Snapshots don't expire.

For very long-term backups of MariaDB, MySQL, and PostgreSQL data, we recommend exporting snapshot data to Amazon S3. If the major version of your DB engine is no longer supported, you can't restore to that version from a snapshot. For more information, see [Exporting DB snapshot data to Amazon S3](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ExportSnapshot.html).

You can create a DB snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

To create a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. In the list of DB instances, choose the DB instance for which you want to take a snapshot.
4. For **Actions**, choose **Take snapshot**.
5. The **Take DB snapshot** window appears.
6. Enter the name of the snapshot in the **Snapshot name** box.


       Take DB snapshot
      

1. Choose **Take snapshot**.

The **Snapshots** page appears, with the new DB snapshot's status shown as Creating. After its status is Available, you can see its creation time.

### Restoring from a DB snapshot

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_RestoreFromSnapshot.html>

Amazon RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. You can create a new DB instance by restoring from a DB snapshot. You provide the name of the DB snapshot to restore from, and then provide a name for the new DB instance that is created from the restore. You can't restore from a DB snapshot to an existing DB instance; a new DB instance is created when you restore.

You can use the restored DB instance as soon as its status is available. The DB instance continues to load data in the background. This is known as lazy loading.

If you access data that hasn't been loaded yet, the DB instance immediately downloads the requested data from Amazon S3, and then continues loading the rest of the data in the background. For more information, see [Amazon EBS snapshots](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html).

To help mitigate the effects of lazy loading on tables to which you require quick access, you can perform operations that involve full-table scans, such as SELECT \*. This allows Amazon RDS to download all of the backed-up table data from S3.

You can restore a DB instance and use a different storage type than the source DB snapshot. In this case, the restoration process is slower because of the additional work required to migrate the data to the new storage type. If you restore to or from magnetic storage, the migration process is the slowest. That's because magnetic storage doesn't have the IOPS capability of Provisioned IOPS or General Purpose (SSD) storage.

To restore a DB instance from a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, for **DB instance identifier**, enter the name for your restored DB instance.
6. Specify other settings, such as allocated storage size.
7. For information about each setting, see [Settings for DB instances](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html#USER_CreateDBInstance.Settings).
8. Choose **Restore DB instance**.

### Deleting a DB snapshot

<https://ecorp.edureka.co/edurekademo/classroom/recording/164/1646/1560141?tab=CourseContent>

You can delete DB snapshots managed by Amazon RDS when you no longer need them.

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. The **Manual snapshots** list appears.
4. Choose the DB snapshot that you want to delete.
5. For **Actions**, choose **Delete snapshot**.
6. Choose **Delete** on the confirmation page.

## Backing up and restoring a Multi-AZ DB cluster

### Creating a Multi-AZ DB cluster snapshot

When you create a Multi-AZ DB cluster snapshot, make sure to identify which Multi-AZ DB cluster you are going to back up, and then give your DB cluster snapshot a name so you can restore from it later. You can also share a Multi-AZ DB cluster snapshot. For instructions, see [Sharing a DB snapshot](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ShareSnapshot.html).

To create a DB cluster snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. In the list, choose the Multi-AZ DB cluster for which you want to take a snapshot.
4. For **Actions**, choose **Take snapshot**.
5. The **Take DB snapshot** window appears.
6. For **Snapshot name**, enter the name of the snapshot.
7. Choose **Take snapshot**.

The **Snapshots** page appears, with the new Multi-AZ DB cluster snapshot's status shown as Creating. After its status is Available, you can see its creation time.

### Restoring from a snapshot to a Multi-AZ DB cluster

You can restore a snapshot to a Multi-AZ DB cluster using the AWS Management Console, the AWS CLI, or the RDS API. You can restore each of these types of snapshots to a Multi-AZ DB cluster:

* A snapshot of a Single-AZ deployment
* A snapshot of a Multi-AZ DB instance deployment with a single DB instance
* A snapshot of a Multi-AZ DB cluster

**Tip**: You can migrate a Single-AZ deployment or a Multi-AZ DB instance deployment to a Multi-AZ DB cluster deployment by restoring a snapshot.

To restore a snapshot to a Multi-AZ DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, in **Availability and durability**, choose **Multi-AZ DB cluster**.


       Multi-AZ DB cluster choice
      

1. For **DB cluster identifier**, enter the name for your restored Multi-AZ DB cluster.
2. For the remaining sections, specify your DB cluster settings. For information about each setting, see [Settings for creating Multi-AZ DB clusters](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/create-multi-az-db-cluster.html#create-multi-az-db-cluster-settings).
3. Choose **Restore DB instance**.

## Working with MySQL read replicas

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_MySQL.Replication.ReadReplicas.html>

* 1. Navigate to the RDS dashboard
  2. Select the DB
  3. Actions -> Create read replica
  4. Select region
  5. Select DB Subnet group. Must be same as the source DB subnet group.
  6. Select AZ in which to launch the read replica. “No preference” will allow AWS RDS to choose the AZ.
  7. Decide if publicly accessible.
  8. Encryption: Only if source is encrypted can the replica b encrypted.
  9. Select the instance class under Instance specs. Preferred: same as source.
  10. Multi-AZ: select yes (or HA) or no based on requirement.
  11. Enter name.
  12. Change port only if necessary.
  13. Copy tags to snapshots, if required.

# Load Balancer

*AWS Solution Architect – Class 9 recording (32:50)*

*Good example:* [*https://www.golinuxcloud.com/aws-application-load-balancer-tutorial/*](https://www.golinuxcloud.com/aws-application-load-balancer-tutorial/)

1. Select region EU – Frankfurt (*Mumbai does not support 3 AZs*)
2. Create a VPC

* 192.168.0.0/16

1. Create 3 Subnets in 3 AZs

* 192.168.1.0/24
* 192.168.2.0/24
* 192.168.3.0/24

1. Create IGW
2. Attach IGW to VPC
3. Create a single Route Table for the 3 public subnets
   1. Create RT for the VPC
   2. Associate the 3 subnets
   3. Create route for internet access
      * 0.0.0.0/0 – IGW created earlier
4. Create 3 Windows EC2 instances in each of the 3 AZs

**Server 01**:

* 1. Windows
  2. t2.micro
  3. New key pair
  4. Select our VPC
  5. Subnet 01
  6. Auto assign public IP: Enable
  7. New security group
     + RDP, HTTP

**Server 02**:

* 1. Windows
  2. t2.micro
  3. select key pair create earlier
  4. Select our VPC
  5. Subnet 02
  6. Auto assign public IP: Enable
  7. Select security group created earlier
     + RDP, HTTP

**Server 03:**

* 1. Windows
  2. t2.micro
  3. select key pair created earlier
  4. Select our VPC
  5. Subnet 03
  6. Auto assign public IP: Enable
  7. Select security group created earlier
     + RDP, HTTP

1. Configure all the servers as web servers

* RDP connect
* d/l .rdp file
* get password from .pem file
* connect
* Server manager
* Add roles and features
* Role-based or feature-based installation
* Select server from server pool
* Web server IIS -> Add features
* Next -> Next -> Next -> Install
* Go to c:\inetpub\wwwroot
* Create a file named index.html with contents on each server with server number:

This is load balancer demo - SERVER-nn !!

1. Configure Load balancer

* EC2 -> Instances -> Load balancers
* Application Load Balancer
* Name
* Internet facing
* IP Address type -> IPv4
* Select our VPC
* Select the 3 subnets
* Select our security group and remove the default one
* Listeners and routing -> Create target group (TG)
* Instances
* Target group name
* HTTP -> 80
* Our VPC
* HTTP1
* Health checks -> Advanced health check settings
* Threshold – 2
* Unhealthy threshold – 2
* Timeout – 5
* Interval – 10
* So, 2 times (threshold) every 10 seconds (interval) if check successful, then marks as Healthy
* Same for unhealthy
* NEXT
* Select all servers
* Include as pending below
* Create Target Group
* Close TG window and go back to LB window
* Refresh TG box
* Select the TG created
* Create Load Balancer
* View LB
* Listeners -> View TG
* Refresh servers till “Healthy”
* View LB -> Refresh till Active
* LB done!
* Now check if servers working with LB
* Go to LB -> Details
* Note/copy DNS name url (endpoint to access application via LB)
* Open browser
* Paste url
* Refresh -> switches between servers
* **Stickiness**
* Go to LB -> Listeners -> TG -> Attributes
* Stickiness -> OFF
* Edit
* Enable Stickiness
* Save changes
* Back to application browser
* Refresh, refresh, refresh: Same server (takes a few seconds to reflect)
* Turn stickiness off and try again (takes a few seconds to reflect)

**Note**: Do not delete as we will use this LB in Auto Scaling demo.

# Auto Scaling

<https://www.golinuxcloud.com/aws-autoscaling-tutorial/>

According to what is mentioned on the official website of [AWS](https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html).

*An Auto Scaling group contains a collection of Amazon EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management.*

* An Auto Scaling group is the place where you define the logic for scaling up and scaling down.
* It has all the rules and policies that govern how the EC2 instances will be terminated or started.
* Auto Scaling groups are the collection of all the EC2 servers running together as a group and dynamically going up or down as per your definitions.
* When you create an Auto Scaling group, first you need to provide the launch configuration that has the details of the instance type, and then you need to choose the scaling plan or scaling policy.

## Different AWS AutoScaling Policy

First, let's look at the different scaling policies that can be applied to the autoscaling groups.

### Dynamic Scaling

* The biggest advantage of AutoScaling is the dynamic scaling of resources based on the demand.
* There is no limit to the number of servers you can scale up to.
* You can scale up from two servers to hundreds or thousands or tens of thousands of servers almost instantly.
* Using AWS Auto Scaling policy you can make sure that your application always performs optimally and gets additional horsepower in terms of CPU and other resources whenever needed. You are able to provision them in real time.

### Predictive scaling

* AWS Auto Scaling is now integrated with machine learning (ML), and by using ML Auto Scaling, you can automatically scale your compute capacity in advance based on predicted increase in demand.
* The way it works is AWS AutoScaling Policy collects the data from your actual usage of EC2 and then uses the machine learning models to predict your daily and weekly expected traffic.
* The data is evaluated every 24 hours to create a forecast for the next 48 hours.

### Scheduled scaling

* If your traffic is predictable and you know that you are going to have an increase in traffic during certain hours, you can have a scaling policy as per the schedule.
* For example, your application may have heaviest usage during the day and hardly any activity at night.
* You can scale the application to have more web servers during the day and scale down during the night.
* To create an AWS AutoScaling policy for scheduled scaling, you need to create a scheduled action that tells the Auto Scaling group to perform the scaling action at the specified time.

## Differences between step scaling policies and simple scaling policies

Step scaling policies and simple scaling policies are two of the dynamic scaling options available for you to use. Both require you to create CloudWatch alarms for the scaling policies. Both require you to specify the high and low thresholds for the alarms. Both require you to define whether to add or remove instances, and how many, or set the group to an exact size.

### Simple Scaling

Simple scaling relies on a metric as a basis for scaling. For example, you can set a CloudWatch alarm to have a CPU Utilization threshold of 80%, and then set the scaling policy to add 20% more capacity to your Auto Scaling group by launching new instances. Accordingly, you can also set a CloudWatch alarm to have a CPU utilization threshold of 30%. When the threshold is met, the Auto Scaling group will remove 20% of its capacity by terminating EC2 instances.

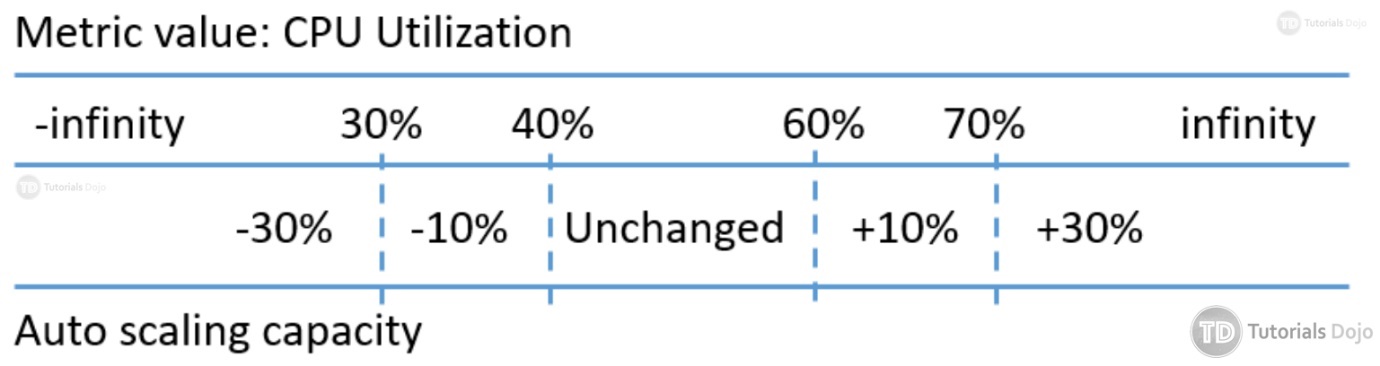
When EC2 Auto Scaling was first introduced, this was the only scaling policy supported. It does not provide any fine-grained control to scaling in and scaling out.

### Step Scaling

Step Scaling further improves the features of simple scaling. Step scaling applies “step adjustments” which means you can set multiple actions to vary the scaling depending on the size of the alarm breach.

When a scaling event happens on simple scaling, the policy must wait for the health checks to complete and the cooldown to expire before responding to an additional alarm. This causes a delay in increasing capacity especially when there is a sudden surge of traffic on your application. With step scaling, the policy can continue to respond to additional alarms even in the middle of the scaling event.

Here is an example that shows how step scaling works:



In this example, the Auto Scaling group maintains its size when the CPU utilization is between 40% and 60%. When the CPU utilization is greater than or equal to 60% but less than 70%, the Auto Scaling group increases its capacity by an additional 10%. When the utilization is greater than 70%, another step in scaling is done and the capacity is increased by an additional 30%. On the other hand, when the overall CPU utilization is less than or equal to 40% but greater than 30%, the Auto Scaling group decreases the capacity by 10%. And if utilization further dips below 30%, the Auto Scaling group removes 30% of the current capacity.

This effectively provides multiple steps in scaling policies that can be used to fine-tune your Auto Scaling group response to dynamically changing workload.

The main difference between the policy types is the step adjustments that you get with step scaling policies. When step adjustments are applied, and they increase or decrease the current capacity of your Auto Scaling group, the adjustments vary based on the size of the alarm breach.

### Summary

In most cases, step scaling policies are a better choice than simple scaling policies, even if you have only a single scaling adjustment.

The main issue with simple scaling is that after a scaling activity is started, the policy must wait for the scaling activity or health check replacement to complete and the [cooldown period](https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-scaling-cooldowns.html) to end before responding to additional alarms. Cooldown periods help to prevent the initiation of additional scaling activities before the effects of previous activities are visible.

In contrast, with step scaling the policy can continue to respond to additional alarms, even while a scaling activity or health check replacement is in progress. Therefore, all alarms that are breached are evaluated by Amazon EC2 Auto Scaling as it receives the alarm messages.

Amazon EC2 Auto Scaling originally supported only simple scaling policies. If you created your scaling policy before target tracking and step policies were introduced, your policy is treated as a simple scaling policy.

## Demo: Configuration of an AutoScaling Group

We will be using Dynamic Scaling as our scaling policy for the tutorial. So, let's dive deep into the process and get our hands dirty!

1. Log in to your AWS account and click on **EC2** under **Services** tab. Now, from the left pane, click on **Auto Scaling Groups** and you will see the following screen.
2. Now click on **Create Auto Scaling Group**. The configuration of Auto Scaling Groups involves seven steps. We will see all of them one by one.

### Step-1: Choose launch template or configuration

1. Here, first, we will provide the name of our AutoScaling Group. For now, we have set it to **My-Test-ASG**. Now we will select a launch template that contains the instance-level settings such as the Amazon Machine Image, instance type, key pair, and security groups. Since we don't have any launch template preconfigured, we will create one.
2. After clicking on **Create a launch template** we will see the following screen. Firstly, we will provide the name of our launch template.
3. Then we will specify the **Amazon Machine Image type**. Here we are selecting it to be **Amazon Linux 2 AMI (HVM), SSD Volume Type**. Then we will select the**Instance type**. Here we are selecting it to **t2.micro**. Note that, it's free tier eligible.
4. Now we will select our existing **Key Pair**. You can also create a new Key Pair.
5. In the **Network Settings**, we will select the **Networking Platform** to be **Virtual Private Cloud** (*if only* **subnets** *are visible, select “****Do not include in template****”*).Then we will select a **Security Group**. Since we already have a Security Group preconfigured, we will select that (ssh-http-https).
6. Then under **Advanced details**, we will set the user data to the following.

#!/bin/bash

yum update -y

yum install -y httpd

systemctl start httpd

systemctl enable httpd

echo "<h1> Hello World from $(hostname -f)<a/h1>" > /var/www/html/index.html

1. Now click on **Create Launch Template.**
2. After creating the launch template, go back to the **Choose launch template or configuration** window, refresh the list of launch templates and it can be seen in the list of available launch templates. Select the created Launch Template and click **Next**.

### Step-2: Choose instance launch options

1. In the **Network**, we will select the default **VPC** in which our Auto Scaling Group will reside. Then we will define which **Availability zones** and **subnets** your Auto Scaling Group can use in the chosen VPC.

### Step-3: Configure advanced options (optional)

1. Here, we will specify that whether we want to use a load balancer or not, and if yes then whether we want to create a new load balancer or choose from an existing load balancer. Since we have a preconfigured **Application Load Balancer** available, for now, we will use that. In the upcoming blog, we will see that how we can create an Application Load Balancer from scratch. Now we will select a **Target Group** from our existing load balancer Target Groups.
2. In Auto Scaling, EC2 instances are automatically replaced if they fail health checks. If using a load balancer, which in our case we are, then we can also enable ELB health check.

### Step-4: Configure group size and scaling policies

1. Here we will set our Group size. For now, we have set the values of **Desired Capacity** and **Minimum Capacity** to **1** and **Maximum** **Capacity** to **3**. Our group size will change according to the traffic received later on. This is the power of Auto Scaling Groups.
2. Now we will select **Target tracking scaling policy** in which we will choose the desired outcome and leave it to the scaling policy to add and remove EC2 instances as needed to achieve that outcome. Here we will select the metric type to **Average CPU utilization** and set the **target value** to **40** which means we don't want any of the instances to run on more than 40% of the CPU utilization.

### Step-5: Add notifications (optional)

1. Here we can add notifications so that they can be sent to SNS topics whenever an Auto Scaling Group launches or terminates an EC2 instance. For now, we are leaving this step since it's optional.

### Step-6: Add tags (optional)

1. Here we can add tags so that they can be used to search, filter, and track our Auto Scaling group. For now, we are not adding any tag.

### Step-7: Review the AWS AutoScaling Policy configuration

1. Now the final step is to review all the details configured. With this, we have come to the end of the creation of our Auto Scaling Group.

### Verify and Monitor AWS AutoScaling Policy Behaviour

1. As soon as we create our AutoScaling group, in the Activity history, we will find some action happening. Here we can see that a new instance has been launched. The newly created EC2 instance can also be viewed after clicking on instances from the left pane. The capacity of Auto Scaling Group has increased from 0 to 1.
2. Now if we look at the **DNS name** of Application Load Balancer and enter it in the browser, we can see that our EC2 instance is working.
3. Now it's time to have some fun. We will increase the CPU utilization of our EC2 instance. As soon as the utilization increases to more than 40% we will see some action happening. First, we will go to our EC2 instance and then click on **Connect**.
4. Again we will click on **Connect**.
5. Now in the console, we will enter the following command.

$ sudo amazon-linux-extras install epel -y

1. Then we will install stress through the following command.

$ sudo yum install stress -y

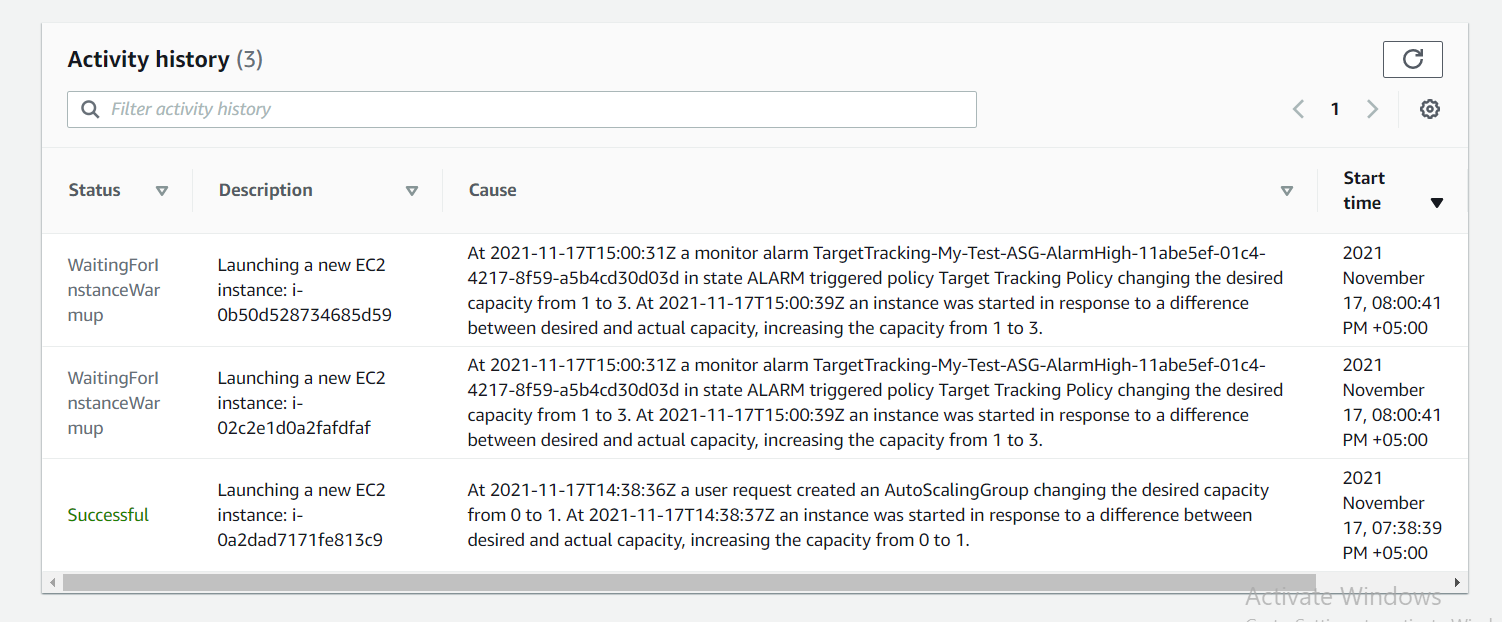
1. After that, we will type in the following command in order to increase CPU utilization.

$ stress -c 4

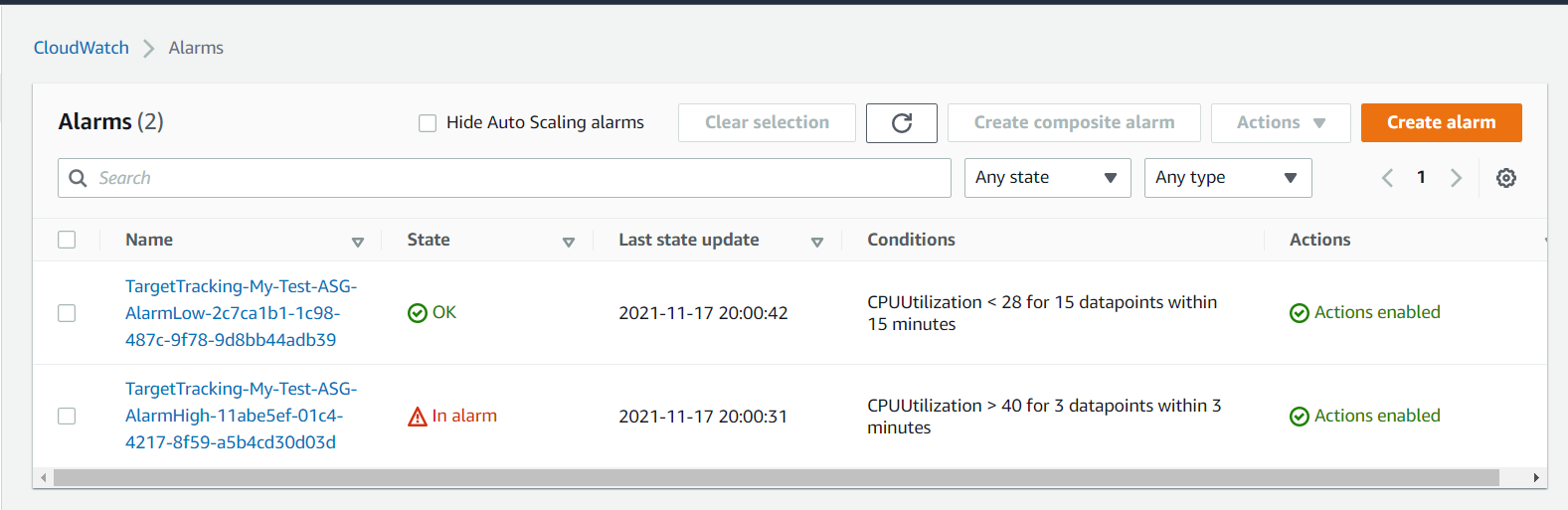
***Alternatively***:

$ while true; do /bin/true; done

1. After a while, in the **Activity history**, we will see that more instances are created in our Auto Scaling group and the capacity has been increased from 1 to 3 since we increased the CPU utilization of our first instance.



1. Can also see new instances being created on EC2 instances dashboard.
2. In the CloudWatch monitoring, we can view the CPU utilization skyrocketing.
3. Now if we go to **CloudWatch Alarms**, they can be found under Services tab, we can clearly see that there are two alarms created automatically by our Target Tracking policy in which one is **AlarmLow** and the other is **AlarmHigh**. It can be seen that right now the AlarmHigh is in the **InAlarm** state since the CPU utilization has been increased.



1. Now let's revert to our previous conditions. For that, we will reboot all the three instances just to make sure that the stress command is killed.
2. After that in the **Activity history** we can clearly see that the capacity is first decreased from 3 to 2 and then 2 to 1. This is the magic of AutoScaling Groups. **This takes time**.
3. In the **CloudWatch Alarms**, now we can see that AlarmLow is in the **InAlarm** state.
4. Terminate all instances. In the, **Activity history** we can see a new instance is provisioned. Check the new instance on **EC2 Instances** dashboard.

## Scaling In / Down

<https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-simple-step.html>

1. Go to **EC2 -> Auto Scaling Groups** and select your Auto Scaling Group.
2. On the **Automatic scaling** tab, in **Dynamic scaling policies**, choose **Create dynamic scaling policy**.
3. To define a policy for scale out (increase capacity), do the following:
4. For **Policy type**, choose **Step scaling**.
5. 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 [Create CloudWatch alarms for the metric high and low thresholds (console)](https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-simple-step.html#policy-creating-alarm-console) to create an alarm that monitors CPU utilization. Set the alarm threshold to less than or equal to 40 percent.
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, 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.

1. 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.
2. Choose **Create**.

# Elastic Block Store (EBS) and Volumes

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html>

## Features of Amazon EBS

* You create an EBS volume in a specific Availability Zone, and then attach it to an instance in that same Availability Zone. To make a volume available outside of the Availability Zone, you can create a snapshot and restore that snapshot to a new volume anywhere in that Region. You can copy snapshots to other Regions and then restore them to new volumes there, making it easier to leverage multiple AWS Regions for geographical expansion, data center migration, and disaster recovery.
* Amazon EBS provides the following volume types: General Purpose SSD, Provisioned IOPS SSD, Throughput Optimized HDD, and Cold HDD. For more information, see [EBS volume types](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html).
* The following is a summary of performance and use cases for each volume type.
  + General Purpose SSD volumes (gp2 and gp3) balance price and performance for a wide variety of transactional workloads. These volumes are ideal for use cases such as boot volumes, medium-size single instance databases, and development and test environments.
  + Provisioned IOPS SSD volumes (io1 and io2) are designed to meet the needs of I/O-intensive workloads that are sensitive to storage performance and consistency. They provide a consistent IOPS rate that you specify when you create the volume. This enables you to predictably scale to tens of thousands of IOPS per instance. Additionally, io2 volumes provide the highest levels of volume durability.
  + Throughput Optimized HDD volumes (st1) provide low-cost magnetic storage that defines performance in terms of throughput rather than IOPS. These volumes are ideal for large, sequential workloads such as Amazon EMR, ETL, data warehouses, and log processing.
  + Cold HDD volumes (sc1) provide low-cost magnetic storage that defines performance in terms of throughput rather than IOPS. These volumes are ideal for large, sequential, cold-data workloads. If you require infrequent access to your data and are looking to save costs, these volumes provides inexpensive block storage.
* You can create your EBS volumes as encrypted volumes, in order to meet a wide range of data-at-rest encryption requirements for regulated/audited data and applications. When you create an encrypted EBS volume and attach it to a supported instance type, data stored at rest on the volume, disk I/O, and snapshots created from the volume are all encrypted. The encryption occurs on the servers that host EC2 instances, providing encryption of data-in-transit from EC2 instances to EBS storage. For more information, see [Amazon EBS encryption](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html).
* You can create point-in-time snapshots of EBS volumes, which are persisted to Amazon S3. Snapshots protect data for long-term durability, and they can be used as the starting point for new EBS volumes. The same snapshot can be used to create as many volumes as needed. These snapshots can be copied across AWS Regions. For more information, see [Amazon EBS snapshots](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html).
* Performance metrics, such as bandwidth, throughput, latency, and average queue length, are available through the AWS Management Console. These metrics, provided by Amazon CloudWatch, allow you to monitor the performance of your volumes to make sure that you are providing enough performance for your applications without paying for resources you don't need. For more information, see [Amazon EBS volume performance on Linux instances](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSPerformance.html).

## Amazon EBS Volumes

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes.html>

An Amazon EBS volume is a durable, block-level storage device that you can attach to your instances. After you attach a volume to an instance, you can use it as you would use a physical hard drive. EBS volumes are flexible. For current-generation volumes attached to current-generation instance types, you can dynamically increase size, modify the provisioned IOPS capacity, and change volume type on live production volumes.

You can use EBS volumes as primary storage for data that requires frequent updates, such as the system drive for an instance or storage for a database application. You can also use them for throughput-intensive applications that perform continuous disk scans. EBS volumes persist independently from the running life of an EC2 instance.

You can attach multiple EBS volumes to a single instance. The volume and instance must be in the same Availability Zone. Depending on the volume and instance types, you can use [Multi-Attach](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes-multi.html) to mount a volume to multiple instances at the same time.

Amazon EBS provides the following volume types: General Purpose SSD (gp2 and gp3), Provisioned IOPS SSD (io1 and io2), Throughput Optimized HDD (st1), Cold HDD (sc1), and Magnetic (standard). They differ in performance characteristics and price, allowing you to tailor your storage performance and cost to the needs of your applications. For more information, see [Amazon EBS volume types](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html).

Your account has a limit on the number of EBS volumes that you can use, and the total storage available to you. For more information about these limits, and how to request an increase in your limits, see [Amazon EC2 service quotas](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-resource-limits.html) (<https://eu-north-1.console.aws.amazon.com/servicequotas/home/services/ebs/quotas>).

### View your current limits

You can view and manage your limits for each Region using the Amazon EC2 console or the Service Quotas console.

To view your current limits using the Amazon EC2 console

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a Region.
3. From the navigation pane, choose **Limits**.
4. Locate the resource in the list. You can use the search fields to filter the list by resource name or resource group. The **Current limit** column displays the current maximum for the resource for your account.

## Amazon EBS volume types

Amazon EBS provides the following volume types, which differ in performance characteristics and price, so that you can tailor your storage performance and cost to the needs of your applications.

**Volume types**

* [Solid state drive (SSD) volumes](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#vol-type-ssd)
* [Hard disk drive (HDD) volumes](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#vol-type-hdd)
* [Previous generation volumes](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#vol-type-prev)

### Solid state drive (SSD) volumes

SSD-backed volumes are optimized for transactional workloads involving frequent read/write operations with small I/O size, where the dominant performance attribute is IOPS. SSD-backed volume types include **General Purpose SSD** and **Provisioned IOPS SSD**.

### Hard disk drive (HDD) volumes

HDD-backed volumes are optimized for large streaming workloads where the dominant performance attribute is throughput. HDD volume types include **Throughput Optimized HDD** and **Cold HDD**.

### Previous generation volumes

Magnetic (standard) volumes are previous generation volumes that are backed by magentic drives. They are suited for workloads with small datasets where data is accessed infrequently and performance is not of primary importance. These volumes deliver approximately 100 IOPS on average, with burst capability of up to hundreds of IOPS, and they can range in size from 1 GiB to 1 TiB.

## Amazon EBS snapshots

You can back up the data on your Amazon EBS volumes to Amazon S3 by taking point-in-time snapshots. Snapshots are incremental backups, which means that only the blocks on the device that have changed after your most recent snapshot are saved. This minimizes the time required to create the snapshot and saves on storage costs by not duplicating data. Each snapshot contains all of the information that is needed to restore your data (from the moment when the snapshot was taken) to a new EBS volume.

When you create an EBS volume based on a snapshot, the new volume begins as an exact replica of the original volume that was used to create the snapshot. The replicated volume loads data in the background so that you can begin using it immediately. If you access data that hasn't been loaded yet, the volume immediately downloads the requested data from Amazon S3, and then continues loading the rest of the volume's data in the background. For more information, see [Create Amazon EBS snapshots](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-creating-snapshot.html).

When you delete a snapshot, only the data unique to that snapshot is removed.

## Amazon EBS vs. Ephemeral Storage

**Ephemeral**: Lasting for a markedly brief time. Having a short lifespan or a short annual period of aboveground growth. Used especially of plants.

<http://www.awsbeginner.com/amazon-ebs-vs-ephemeral-storage/>

EC2 Instances primarily utilize storage known as [EBS Volumes](http://34.205.144.76/ebs-volumes-101/).  But this isn’t the only type of storage that is available.  There’s another type, known as **Instance Store**, or **Ephemeral Storage**.

Instance Store was first used by Amazon, until they shifted to Elastic Block Store.  There are several **differences** between **EBS** and **Ephemeral** **Storage** that may be worth understanding for us as Cloud Engineers.

| **Elastic Block Store (EBS)** | **Ephemeral Storage (Instance Store)** |
| --- | --- |
| The DeFacto standard Storage Option for EC2 | A legacy option that's still available for EC2 storage. |
| Provisioned from Snapshots created from Elastic Block Storage | Provisioned from templates stored in Amazon S3 |
| Can be attached/deattach/reattached to an EC2 instance | Cannot be detached/reattached to an EC2 instance |
| Able to restart the EC2. instance. | Unable to Restart the Instance once running. Only Option is Terminate |
| Able to both start and stop EBS backed EC2 instances | Unable to stop the EC2 once it starts running. |
| Can attach as many EBS volumes to an EC2 as is desired. | Cannot only attach a max of ONE additional Instance Store to the EC2 |

The [AWS Documentation](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html) mentions that Amazon Instance Store is designed for temporary storage for EC2.  Things like [Buffer](https://www.computerhope.com/jargon/b/buffer.htm), [Cache,](https://www.computerhope.com/jargon/c/cache.htm), in Raid Arrays, or Temporary Backups. are some possible use cases for instance store.

I was surprised to see this storage option besides EBS, because EBS is the default.  EBS is is the DeFacto standard when launching an EC2 instance.

It’s actually pretty tricky to figure out where you can actually use ephemeral storage when launching EC2 instances.  For example, if you try to launch a **t2.micro** EC2, or even most of the other instance types, you will not be able to attach ephemeral storage.

But if you launch the [**m3.medium**](http://34.205.144.76/ec2-instance-types/) EC2, then you are able to add ONE instance store.  And for the **g2.8xlarge**, it allows you to add TWO additional instance stores.  And if you create a new AMI image from a launched EC2 instance, then you can attach as many Instance store volumes as are available (24 total).

## Demo: Increase the size of an Amazon EBS volume on an EC2 instance

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/modify-ebs-volume-on-instance.html>

This tutorial shows you how to launch an Amazon EC2 instance and add an Amazon Elastic Block Store (Amazon EBS) volume using the EC2 launch instance wizard. The added volume must be formatted with a file system and mounted to be available for use. Suppose, after using this configuration for six months, your added volume is full. This tutorial shows you how to double the size of the added volume, and extend the file system.

Consider the following as you complete the steps in this tutorial:

* **Two volumes** – After you launch your Amazon EC2 instance you have two volumes—a root volume and an added volume. To differentiate between these volumes, the added volume will be called a data volume in this tutorial.
* **Data volume availability** – To make the data volume available, you need to format and mount the data volume.
* **File system** – To take advantage of the increased size of the data volume, the file system needs to be extended.

### Step 1: Launch an EC2 instance with an added EBS volume

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the **EC2 Dashboard**, choose **Launch instance**.
3. Under **Name and tags**, for **Name**, enter **tutorial-volumes**.
4. Under **Application and OS Images**, select **Quick Start**, and verify the following defaults:
   * **Amazon Linux** is selected as the OS for your instance.
   * Under **Amazon Machine Image (AMI)**, **Amazon Linux 2 AMI** is selected.
5. Under **Instance type**, select t3.micro.
6. Under **Key pair (login)**, choose your key pair.
7. Under **Network settings**, ensure that **Allow SSH traffic** check box is selected. This allows you to connect to your instance using SSH.
8. Under **Configure storage**, choose **Add new volume**. Ensure that the added **EBS volume** size is **8 GB** and the type is gp3.
   * To ensure that the data volume is deleted upon termination of the instance, choose **Advanced**. Under **Volume 2 (Custom)**, select **Delete on termination**, and select **Yes**.
9. In the **Summary** panel, choose **Launch instance**.
10. Choose **View all instances** to close the confirmation page and return to the Amazon EC2 console.

### Step 2: Make the data volume available for use

The Amazon EC2 instance launched in [Step 1](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/step1-launch-instance-with-ebs-volume.html) has two EBS volumes—a root device and an added storage device (the data volume). Use the following procedure to make the data volume available for use. In this tutorial, you use the XFS file system.

**Note:** The instance type (t3.micro) used in this tutorial is built on the [Nitro System](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html#ec2-nitro-instances), which exposes EBS volumes as NVMe block devices.

To format and mount the data volume

1. In the Amazon EC2 console, in the navigation pane, choose **Instances**, and select **tutorial-volumes**.
2. To connect to your instance, choose **Connect**, ensure **EC2 Instance Connect** is selected, and then choose **Connect**.
3. In the terminal window, use the following command to view your available disk devices and their mount points. The output of **lsblk** removes the /dev/ prefix from full device paths.

[ec2-user ~]$ **sudo lsblk -f**

The following example output shows that there are two devices attached to the instance—the root device /dev/nvme0n1 and the data volume /dev/nvme1n1. In this example, you can see under the column FSTYPE that the data volume does not have a file system.


                        Display of file systems on an EC2 instance storage volume using EC2
                            Instance Connect.
                    

1. To create a file system, use the following command with the data volume name (/dev/nvme1n1).

[ec2-user ~]$ **sudo mkfs -t xfs */dev/nvme1n1***

1. To create a mount point directory for the data volume, use the following command. The mount point is where the volume is located in the file system tree and where you read and write files to after you mount the volume. The following example creates a directory named /data.

[ec2-user ~]$ **sudo mkdir */data***

1. To mount the data volume at the directory you just created, use the following command with the data volume name (/dev/nvme1n1) and the mount point directory name (/data).

[ec2-user ~]$ **sudo mount */dev/nvme1n1 /data***

1. The mount point is not automatically saved after rebooting your instance. To automatically mount the data volume after reboot, see [Automatically mount an attached volume after reboot](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-using-volumes.html).
2. To verify that you have formatted and mounted your data volume, run the following command again.

[ec2-user ~]$ **lsblk -f**

The following example output shows that the data volume /dev/nvme1n1 has an XFS file system and is mounted at the /data mount point.


                        Display of file systems confirming data volume has a file system and is mounted.
                    

### Step 3: Increase the size of the data volume

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/step3-increase-size-of-data-volume.html>

Suppose after six months of using the above configuration you run out of storage space on your data volume. You decide to double the size of your data volume. To do this, first you create a snapshot, and then you increase the size of the data volume. When you create a snapshot, use a description to identify the snapshot. In this tutorial you create a snapshot called tutorial-volumes-backup.

**Note**: When you modify an EBS volume, it goes through a sequence of states—modifying, optimizing, and completed. Keep in mind that the file system cannot be extended until the volume enters the optimizing state.

To increase the size of the data volume

1. Return to the Amazon EC2 console by closing the browser window of EC2 Instance Connect.
2. Create a snapshot of the data volume, in case you need to roll back your changes.
   * In the Amazon EC2 console, in the navigation pane, choose **Instances**, and select **tutorial-volumes**.
   * On the **Storage** tab, under **Block devices** select the **Volume ID** of the data volume.
   * On the **Volumes** detail page, choose **Actions**, and **Create snapshot**.
   * Under **Description**, enter **tutorial-volumes-backup**.
   * Choose **Create snapshot**.
3. To increase the data volume size, in the navigation pane, choose **Instances**, and select **tutorial-volumes**.
4. Under the **Storage** tab, select the **Volume ID** of your data volume.
5. Select the check box for your **Volume ID**, choose **Actions**, and then **Modify volume**.
6. The **Modify volume** screen displays the volume ID and the volume’s current configuration, including type, size, input/output operations per second (IOPS), and throughput. In this tutorial you double the size of the data volume.
   * For **Volume type**, do not change value.
   * For **Size**, change to 16 GB.
   * For **IOPS**, do not change value.
   * For **Throughput**, do not change value.
7. Choose **Modify**, and when prompted for confirmation choose **Modify** again. You are charged for the new volume configuration after volume modification starts. For pricing information, see [Amazon EBS Pricing](http://aws.amazon.com/ebs/pricing/?nc1=h_ls).

**Note**: You must wait at least six hours and ensure the volume is in the in-use or available state before you can modify the volume again.

### Step 4: Extend the file system

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/step4-extend-file-system.html>

Once the data volume enters the optimizing state, you can use file system-specific commands to extend the file system to the new, larger size.

To extend the file system

1. In the Amazon EC2 console, on the EC2 Dashboard, choose **Instances**, and select **tutorial-volumes**.
2. To connect to your instance, choose **Connect**, ensure **EC2 Instance Connect** is selected, and then choose **Connect**.
3. In the terminal window, use the following command to get the size of the file system.

[ec2-user ~]$ **df -hT**

The following example output shows that the file system size of the data volume /dev/nvme1n1 is 8 GB. In the previous procedure, you increased the data volume size to 16 GB. Now you need to extend the file system to take advantage of the added storage.


                       Display of file systems showing data volume file system size has not been increased.
                   

1. Use the following command to extend the XFS file system that is mounted on the /data mount point.

[ec2-user ~]$ **sudo xfs\_growfs -d */data***

1. Use the following command again to verify that the file system has been extended.

[ec2-user ~]$ **df -hT**

The following example output shows that the file system size of /dev/nvme1n1 is the same as the data volume size.


                       Display of file systems showing data volume file system size has been increased.
                   

# Route 53

*AWS Solution Architect – Class 6 presentation*

## What is Amazon Route53?

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. You can use Route 53 to perform three main functions in any combination: domain registration, DNS routing, and health checking.

If you choose to use Route 53 for all three functions, be sure to follow the order below:

### 1. Register domain names

Your website needs a name, such as example.com. Route 53 lets you register a name for your website or web application, known as a domain name.

* For an overview, see [How domain registration works](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-domain-registration.html).
* For a procedure, see [Registering a new domain](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/domain-register.html).
* For a tutorial that takes you through registering a domain and creating a simple website in an Amazon S3 bucket, see [Getting started with Amazon Route 53](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/getting-started.html).

### 2. Route internet traffic to the resources for your domain

When a user opens a web browser and enters your domain name (example.com) or subdomain name (acme.example.com) in the address bar, Route 53 helps connect the browser with your website or web application.

* For an overview, see [How internet traffic is routed to your website or web application](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-dns-service.html).
* For procedures, see [Configuring Amazon Route 53 as your DNS service](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-configuring.html).
* For a procedure on how to route email to Amazon WorkMail, see [Routing traffic to Amazon WorkMail](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-to-workmail.html).

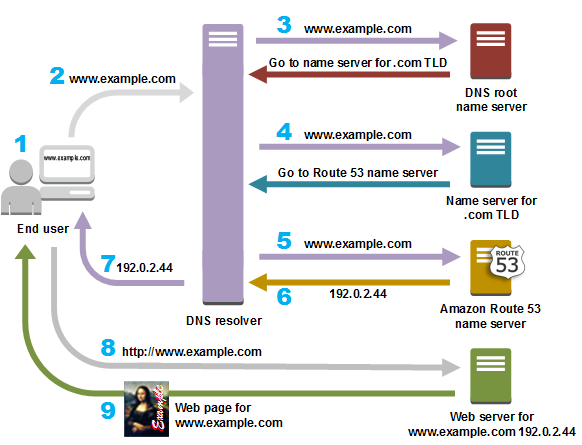
### 3. Check the health of your resources

Route 53 sends automated requests over the internet to a resource, such as a web server, to verify that it's reachable, available, and functional. You also can choose to receive notifications when a resource becomes unavailable and choose to route internet traffic away from unhealthy resources.

* For an overview, see [How Amazon Route 53 checks the health of your resources](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-health-checks.html).
* For procedures, see [Creating Amazon Route 53 health checks and configuring DNS failover](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html).

## How Amazon Route 53 routes traffic for your domain

After you configure Amazon Route 53 to route your internet traffic to your resources, such as web servers or Amazon S3 buckets, here's what happens in just a few milliseconds when someone requests content for www.example.com:



1. A user opens a web browser, enters www.example.com in the address bar, and presses Enter.
2. The request for www.example.com is routed to a DNS resolver, which is typically managed by the user's internet service provider (ISP), such as a cable internet provider, a DSL broadband provider, or a corporate network.
3. The DNS resolver for the ISP forwards the request for www.example.com to a DNS root name server.
4. The DNS resolver forwards the request for www.example.com again, this time to one of the TLD name servers for .com domains. The name server for .com domains responds to the request with the names of the four Route 53 name servers that are associated with the example.com domain.

The DNS resolver caches (stores) the four Route 53 name servers. The next time someone browses to example.com, the resolver skips steps 3 and 4 because it already has the name servers for example.com. The name servers are typically cached for two days.

1. The DNS resolver chooses a Route 53 name server and forwards the request for www.example.com to that name server.
2. The Route 53 name server looks in the example.com hosted zone for the www.example.com record, gets the associated value, such as the IP address for a web server, 192.0.2.44, and returns the IP address to the DNS resolver.
3. The DNS resolver finally has the IP address that the user needs. The resolver returns that value to the web browser.

**Note**: The DNS resolver also caches the IP address for example.com for an amount of time that you specify so that it can respond more quickly the next time someone browses to example.com. For more information, see [time to live (TTL)](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/route-53-concepts.html#route-53-concepts-time-to-live).

1. The web browser sends a request for www.example.com to the IP address that it got from the DNS resolver. This is where your content is, for example, a web server running on an Amazon EC2 instance or an Amazon S3 bucket that's configured as a website endpoint.
2. The web server or other resource at 192.0.2.44 returns the web page for www.example.com to the web browser, and the web browser displays the page.

### Simple routing policy

Use for a single resource that performs a given function for your domain, for example, a web server that serves content for the example.com website.

Simple routing lets you configure standard DNS records, with no special Route 53 routing such as weighted or latency. With simple routing, you typically route traffic to a single resource, for example, to a web server for your website.

You can use simple routing to create records in a private hosted zone.

### Failover routing policy

Use when you want to configure active-passive failover.

You can use failover routing to create records in a private hosted zone.

### Geolocation routing policy

Use when you want to route traffic based on the location of your users.

Geolocation routing lets you choose the resources that serve your traffic based on the geographic location of your users, meaning the location that DNS queries originate from. For example, you might want all queries from Europe to be routed to an ELB load balancer in the Frankfurt region.

When you use geolocation routing, you can localize your content and present some or all of your website in the language of your users. You can also use geolocation routing to restrict distribution of content to only the locations in which you have distribution rights. Another possible use is for balancing load across endpoints in a predictable, easy-to-manage way, so that each user location is consistently routed to the same endpoint.

You can specify geographic locations by continent, by country, or by state in the United States. If you create separate records for overlapping geographic regions—for example, one record for North America and one for Canada—priority goes to the smallest geographic region. This allows you to route some queries for a continent to one resource and to route queries for selected countries on that continent to a different resource. (For a list of the countries on each continent, see [Location](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-values-geo.html#rrsets-values-geo-location).)

Geolocation works by mapping IP addresses to locations. However, some IP addresses aren't mapped to geographic locations, so even if you create geolocation records that cover all seven continents, Amazon Route 53 will receive some DNS queries from locations that it can't identify. You can create a default record that handles both queries from IP addresses that aren't mapped to any location and queries that come from locations that you haven't created geolocation records for. If you don't create a default record, Route 53 returns a "no answer" response for queries from those locations.

You can use geolocation routing to create records in a private hosted zone.

### Geoproximity routing policy

Use when you want to route traffic based on the location of your resources and, optionally, shift traffic from resources in one location to resources in another.

Route 53's Geo DNS support lets you balance load by directing requests to specific endpoints based on the geographic location from which the request originates. Geo DNS makes it possible to customize localized content, such as presenting detail pages in the right language or restricting distribution of content to only the markets you have licensed. Geo DNS also lets you balance load across endpoints in a predictable, easy-to-manage way, ensuring that each end-user location is consistently routed to the same endpoint. Geo DNS provides three levels of geographic granularity: continent, country, and state, and Geo DNS also provides a global record which is served in cases where an end user's location doesn't match any of the specific Geo DNS records you have created. You can also combine Geo DNS with other routing types, such as Latency Based Routing and DNS Failover, using Route 53's Alias feature in order to enable a variety of low-latency and fault-tolerant architectures To learn more, please see the [Route 53 documentation](http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html#routing-policy-geo).

Geoproximity routing lets Amazon Route 53 route traffic to your resources based on the geographic location of your users and your resources. You can also optionally choose to route more traffic or less to a given resource by specifying a value, known as a bias. A bias expands or shrinks the size of the geographic region from which traffic is routed to a resource.

To use geoproximity routing, you must use Route 53 [traffic flow](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/traffic-flow.html). You create geoproximity rules for your resources and specify one of the following values for each rule:

* If you're using AWS resources, the AWS Region that you created the resource in
* If you're using non-AWS resources, the latitude and longitude of the resource

To optionally change the size of the geographic region from which Route 53 routes traffic to a resource, specify the applicable value for the bias:

* To expand the size of the geographic region from which Route 53 routes traffic to a resource, specify a positive integer from 1 to 99 for the bias. Route 53 shrinks the size of adjacent regions.
* To shrink the size of the geographic region from which Route 53 routes traffic to a resource, specify a negative bias of -1 to -99. Route 53 expands the size of adjacent regions.

You cannot use geoproximity routing policy for records in a private hosted zone.

### Latency routing policy (LBR)

Use when you have resources in multiple AWS Regions and you want to route traffic to the region that provides the best latency.

For example, suppose you have ELB load balancers in the US West (Oregon) Region and in the Asia Pacific (Singapore) Region. You created a latency record for each load balancer. Here's what happens when a user in London enters the name of your domain in a browser:

1. DNS routes the query to a Route 53 name server.
2. Route 53 refers to its data on latency between London and the Singapore region and between London and the Oregon region.
3. If latency is lower between the London and Oregon regions, Route 53 responds to the query with the IP address for the Oregon load balancer. If latency is lower between London and the Singapore region, Route 53 responds with the IP address for the Singapore load balancer.

LBR, one of Amazon Route 53’s most requested features, helps you improve your application’s performance for a global audience. LBR works by routing your customers to the AWS endpoint (e.g. EC2 instances, Elastic IPs or ELBs) that provides the fastest experience based on actual performance measurements of the different AWS regions where your application is running.

You can use latency routing to create records in a private hosted zone.

### IP-based routing policy

Use when you want to route traffic based on the location of your users, and have the IP addresses that the traffic originates from.

With the addition of IP-based routing, customers are now additionally empowered to fine-tune their DNS routing approach based on the Classless Inter-Domain Routing (CIDR) block that the query-originating IP address belongs to, allowing them to leverage knowledge of their end user base to optimize performance or network transit costs.

For instance, you can now route end users within certain Internet Service Provider (ISP) networks to specific endpoints such as Content Delivery Networks (CDNs). These ISP-to-CDN mappings might be unique for each customer and based on factors such as business contracts with CDNs or a partner ISP’s network topology. Customers who have developed routing decision maps based on their own analysis and want to holistically apply them to Route 53 are now able to upload IP address prefixes (CIDR blocks) to Route 53, group them into reusable entities called CIDR collections, and associate these collections with one or more Resource Record Sets (RRSets). For customers who want to selectively take advantage of IP-based routing for specific overrides, IP-based routing can also be used in combination with existing routing types such as geolocation routing.

You cannot use IP-based routing policy for records in a private hosted zone.

### Multivalue answer routing policy

Use when you want Route 53 to respond to DNS queries with up to eight healthy records selected at random.

Multivalue answer routing lets you configure Amazon Route 53 to return multiple values, such as IP addresses for your web servers, in response to DNS queries. You can specify multiple values for almost any record, but multivalue answer routing also lets you check the health of each resource, so Route 53 returns only values for healthy resources. It's not a substitute for a load balancer, but the ability to return multiple health-checkable IP addresses is a way to use DNS to improve availability and load balancing.

To route traffic approximately randomly to multiple resources, such as web servers, you create one multivalue answer record for each resource and, optionally, associate a Route 53 health check with each record. Route 53 responds to DNS queries with up to eight healthy records and gives different answers to different DNS resolvers. If a web server becomes unavailable after a resolver caches a response, client software can try another IP address in the response.

You can use multivalue answer routing to create records in a private hosted zone.

### Weighted routing policy

Use to route traffic to multiple resources in proportions that you specify.

Weighted routing lets you associate multiple resources with a single domain name (example.com) or subdomain name (acme.example.com) and choose how much traffic is routed to each resource. This can be useful for a variety of purposes, including load balancing and testing new versions of software.

To configure weighted routing, you create records that have the same name and type for each of your resources. You assign each record a relative weight that corresponds with how much traffic you want to send to each resource. Amazon Route 53 sends traffic to a resource based on the weight that you assign to the record as a proportion of the total weight for all records in the group:


     Formula for how much traffic is routed to a given resource: 
      weight for a specified record / sum of the weights for all records.
    

For example, if you want to send a tiny portion of your traffic to one resource and the rest to another resource, you might specify weights of 1 and 255. The resource with a weight of 1 gets 1/256th of the traffic (1/(1+255)), and the other resource gets 255/256ths (255/(1+255)). You can gradually change the balance by changing the weights. If you want to stop sending traffic to a resource, you can change the weight for that record to 0.

You can use weighted routing to create records in a private hosted zone.

## How Amazon Route 53 checks the health of your resources

<https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-health-checks.html>

Amazon Route 53 health checks monitor the health of your resources such as web servers and email servers. You can optionally configure Amazon CloudWatch alarms for your health checks, so that you receive notification when a resource becomes unavailable.

Here's an overview of how health checking works if you want to be notified when a resource becomes unavailable:


    Conceptual graphic that shows how you configure Route 53 to monitor the health of specified endpoints.
   

1. You create a health check and specify values that define how you want the health check to work, such as the following:

* The IP address or domain name of the endpoint, such as a web server, that you want Route 53 to monitor. (You can also monitor the status of other health checks, or the state of a CloudWatch alarm.)
* The protocol that you want Amazon Route 53 to use to perform the check: HTTP, HTTPS, or TCP.
* How often you want Route 53 to send a request to the endpoint. This is the request interval.
* How many consecutive times the endpoint must fail to respond to requests before Route 53 considers it unhealthy. This is the failure threshold.
* Optionally, how you want to be notified when Route 53 detects that the endpoint is unhealthy. When you configure notification, Route 53 automatically sets a CloudWatch alarm. CloudWatch uses Amazon SNS to notify users that an endpoint is unhealthy.

1. Route 53 starts to send requests to the endpoint at the interval that you specified in the health check.

If the endpoint responds to the requests, Route 53 considers the endpoint to be healthy and takes no action.

1. If the endpoint doesn't respond to a request, Route 53 starts to count the number of consecutive requests that the endpoint doesn't respond to:
   1. If the count reaches the value that you specified for the failure threshold, Route 53 considers the endpoint unhealthy.
   2. If the endpoint starts to respond again before the count reaches the failure threshold, Route 53 resets the count to 0, and CloudWatch doesn't contact you.
2. If Route 53 considers the endpoint unhealthy and if you configured notification for the health check, Route 53 notifies CloudWatch.

If you didn't configure notification, you can still see the status of your Route 53 health checks in the Route 53 console. For more information, see [Monitoring health check status and getting notifications](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/health-checks-monitor-view-status.html).

1. If you configured notification for the health check, CloudWatch triggers an alarm and uses Amazon SNS to send notification to the specified recipients.

## Configure Health Check for EC2 Instances

<https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/getting-started-s3.html#getting-started-find-domain-name>

Route 53 considers a new health check to be healthy until there's enough data to determine the actual status, healthy or unhealthy.

If you chose the option to invert the health check status, Route 53 considers a new health check to be unhealthy until there's enough data.

**Steps**:

1. Region US – N. Virginia
2. Create 3 Linux instances in default VPC

* Create 1 instance with option “Number of instances: 3”
* Amazon Linux 2 Kernel 5.10 AMI 2.0.20230320.0 x86\_64 HVM gp2
* Default VPC
* Security Group: Allow SSH, HTTPS, HTTP
* Launch instance
* Change names, if required

1. Create R53 health check

* R53 -> Create Health Check
* Name
* Endpoint
* IP Address
* Protocol: HTTP
* IP Address: of instance 1
* Hostname: of instance 1
* Port: 80
* Path: AMI name of instance 1 (amzn2-ami-kernel-5.10-hvm-2.0.20230320.0-x86\_64-gp2)
* Advanced config: default
* Next
* Get notified: No
* Create health check
* Repeat steps for instances 2 & 3
* Check health of all 3 instances
* Delete health checks and instances (keep if required for CloudWatch demo)

# AWS CloudWatch – Monitoring

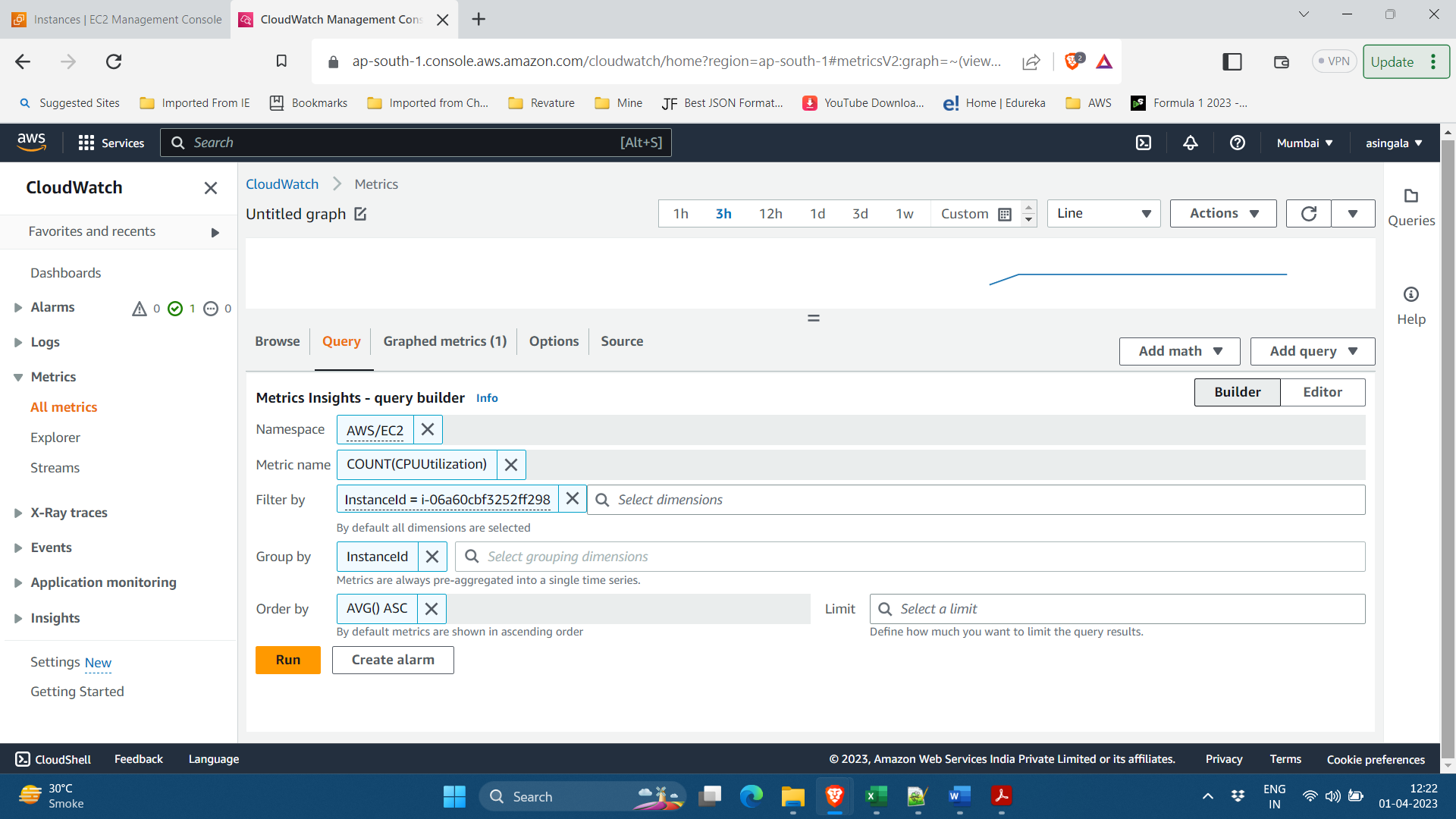
*AWS Solution Architect – Module 9 Demo 1*

**Note**: Can also create new alarm by creating an EC2 instance and then from Monitoring or Status Checks tabs of the instance.

1. Region US – N. Virginia
2. Create 1 or more Linux AMI 2 instances in default VPC
3. Create Route53 health checks (HC)
4. On Ec2 instance, select Monitoring tab
5. Create a CloudWatch alarm from the CloudWatch console
6. Create alarms -> Create alarm
7. Select metric
8. Browse -> EC2 -> Per-instance metrics
9. Select CPUUtilization for the R53 HC (s)
10. Select metrics
11. Metric name: CPUUtilization
12. Instance id
13. Statistic: Average
14. Period: 5 minutes
15. Static
16. Lower/Equal (<=) 100
17. Next
18. Create new SNS topic

* Check email and confirm subscription

1. Email id
2. Create Topic
3. Select existing topic
4. Next
5. Alarm name
6. Review -> Create Alarm
7. Check if “Actions enabled”
8. Select alarm
9. Select Actions tab
10. Now, run a CloudWatch Metrics Insights query using the CloudWatch console
11. In the navigation pane on the left, select Metrics -> All Metrics -> Browse
12. “View automatic dashboard” under EC2
13. In the navigation pane on the left, select Metrics -> All Metrics -> Query
14. Namespace: AWS/EC2
15. Metric name: COUNT(CPUUtilization)
16. Filter by: InstanceId = <EC2 instance id>
17. Group by: InstanceId
18. Order by: AVG() ASC
19. Run
20. In the navigation pane on the left, select Metrics -> All Metrics -> Graphed Metrics
21. Remove the query created by clicking on the “X” under Actions
22. Alarms -> select alarm -> Actions -> Delete



## Dashboard

* Create a new dashboard
* Show around
* Share publicly (to anyone)

## Monitor EBS

* Select an instance.
* Note it’s volume id.
* Create an alarm

## Billing Alarm

* From CloudWatch dashboard, switch to N. Virginia (us-east-1) region.
* Select “Billing” alarm from left panel.
* Create a new alarm.
* Show “Billing dashboard” from user menu on top-right.

### Creating a billing alarm

**Important**: Before you create a billing alarm, you must set your Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges. You also must enable billing alerts for your account or in the management/payer account (if you are using consolidated billing).

In this procedure, you create an alarm that sends a notification when your estimated charges for AWS exceed a defined threshold.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**. In **Browse**, choose **Billing**, and then choose **Total Estimated Charge**.

**Note:** If you dont't see the **Billing**/**Total Estimated Charge** metric, enable billing alerts, and change your Region to US East (N. Virginia). For more information, see [Enabling billing alerts](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics).

1. Select the box for the **EstimatedCharges** metric, and then choose **Select metric**.
2. For **Statistic**, choose **Maximum**.
3. For **Period**, choose **6 hours**.
4. For **Threshold type**, choose **Static**.
5. For **Whenever EstimatedCharges is . . .**, choose **Greater**.
6. For **than . . .**, define a threshold value that triggers your alarm (for example, **200** USD).

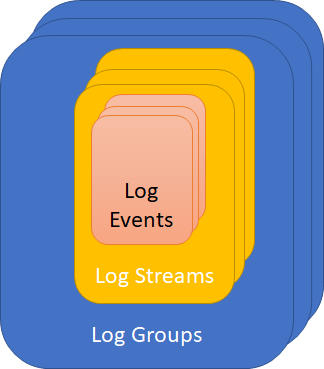
The **EstimatedCharges** metric values are only in US dollars (USD), and the currency conversion is provided by Amazon Services LLC. For more information, see [What is AWS Billing?](https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-what-is.html).

**Note:** After you define a threshold value, the preview graph displays your estimated charges for the current month.

1. In Additional Configuration, do the following:
   * For Datapoints to alarm, specify 1 out of 1.
   * For Missing data treatment, choose Treat missing data as missing.
2. Choose Next.
3. Under Notification, specify an Amazon SNS topic to be notified when your alarm is in the ALARM state. You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose Add notification.
4. Choose Next.
5. Under Name and description, enter a name for your alarm.
   * (Optional) Enter a description of your alarm.
6. Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

## Amazon CloudWatch Logs concepts

The terminology and concepts that are central to your understanding and use of CloudWatch Logs are described below.



### Log events

A log event is a record of some activity recorded by the application or resource being monitored. The log event record that CloudWatch Logs understands contains two properties: the timestamp of when the event occurred, and the raw event message. Event messages must be UTF-8 encoded.

### Log streams

A log stream is a sequence of log events that share the same source. More specifically, a log stream is generally intended to represent the sequence of events coming from the application instance or resource being monitored. For example, a log stream may be associated with an Apache access log on a specific host. When you no longer need a log stream, you can delete it using the [aws logs delete-log-stream](https://docs.aws.amazon.com/cli/latest/reference/logs/delete-log-stream.html) command.

### Log groups

Log groups define groups of log streams that share the same retention, monitoring, and access control settings. Each log stream has to belong to one log group. For example, if you have a separate log stream for the Apache access logs from each host, you could group those log streams into a single log group called MyWebsite.com/Apache/access\_log.

There is no limit on the number of log streams that can belong to one log group.

### Metric filters

You can use metric filters to extract metric observations from ingested events and transform them to data points in a CloudWatch metric. Metric filters are assigned to log groups, and all of the filters assigned to a log group are applied to their log streams.

#### Examples

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/MonitoringPolicyExamples.html>

#### Example: Count log events

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CountingLogEventsExample.html>

The simplest type of log event monitoring is to count the number of log events that occur. You might want to do this to keep a count of all events, to create a "heartbeat" style monitor or just to practice creating metric filters.

In the following CLI example, a metric filter called MyAppAccessCount is applied to the log group MyApp/access.log to create the metric EventCount in the CloudWatch namespace MyNamespace. The filter is configured to match any log event content and to increment the metric by "1".

To create a metric filter **EventCount** using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of a log group.
4. Choose Actions, **Create metric filter**.
5. Leave **Filter Pattern** and **Select Log Data to Test** blank.
6. Choose **Next**, and then for **Filter Name**, type **EventCount**.
7. Under **Metric Details**, for **Metric Namespace**, type **MyNameSpace**.
8. For **Metric Name**, type **MyAppEventCount**.
9. Confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every log event.
10. For **Default Value** enter 0, and then choose **Next**. Specifying a default value ensures that data is reported even during periods when no log events occur, preventing spotty metrics where data sometimes does not exist.
11. Choose **Create metric filter**.

To create a metric filter **EventCount** using the AWS CLI

At a command prompt, run the following command:

**aws logs put-metric-filter \**

**--log-group-name MyApp/access.log \**

**--filter-name EventCount \**

**--filter-pattern " " \**

**--metric-transformations \**

**metricName=MyAppEventCount,metricNamespace=MyNamespace,metricValue=1,defaultValue=0**

You can test this new policy by posting any event data. You should see data points published to the metric MyAppAccessEventCount.

To post event to **EventCount** data using the AWS CLI

At a command prompt, run the following command:

**aws logs put-log-events \**

**--log-group-name MyApp/access.log --log-stream-name TestStream1 \**

**--log-events \**

**timestamp=1394793518000,message="Test event 1" \**

**timestamp=1394793518000,message="Test event 2" \**

**timestamp=1394793528000,message="This message also contains an Error"**

Once metrics filter is created, we can create an alarm on that or view graph. So, if in a log event, a particular term etc. is observed more than 5 times, trigger an alarm, send notification to SNS, which will send an email to someone.

#### Example: Count occurrences of a term

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CountOccurrencesExample.html>

Log events frequently include important messages that you want to count, maybe about the success or failure of operations. For example, an error may occur and be recorded to a log file if a given operation fails. You may want to monitor these entries to understand the trend of your errors.

In the example below, a metric filter is created to monitor for the term Error. The policy has been created and added to the log group **MyApp/message.log**. CloudWatch Logs publishes a data point to the CloudWatch custom metric ErrorCount in the **MyApp/message.log** namespace with a value of "1" for every event containing Error. If no event contains the word Error, then a value of 0 is published. When graphing this data in the CloudWatch console, be sure to use the sum statistic.

After you create a metric filter, you can view the metric in the CloudWatch console. When you are selecting the metric to view, select the metric namespace that matches the log group name. For more information, see [Viewing Available Metrics](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/viewing_metrics_with_cloudwatch.html).

To create a metric filter **MyAppErrorCount** using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the log group.
4. Choose **Actions**, **Create metric filter**.
5. For **Filter Pattern**, enter **Error**.

**Note:** All entries in **Filter Pattern** are case-sensitive.

1. (Optional) To test your filter pattern, under **Test Pattern**, enter one or more log events to use to test the pattern. Each log event must be within one line, because line breaks are used to separate log events in the **Log event messages** box.
2. Choose **Next**, and then on the **Assign metric** page, for **Filter Name**, type **MyAppErrorCount**.
3. Under **Metric Details**, for **Metric Namespace**, type **MyNameSpace**.
4. For **Metric Name**, type **ErrorCount**.
5. Confirm that **Metric Value** is 1. This specifies that the count is incremented by 1 for every log event containing "Error".
6. For **Default Value** type 0, and then choose **Next**.
7. Choose Create metric filter.

To create a metric filter **MyAppErrorCount** using the AWS CLI

At a command prompt, run the following command:

**aws logs put-metric-filter \**

**--log-group-name MyApp/message.log \**

**--filter-name MyAppErrorCount \**

**--filter-pattern "Error" \**

**--metric-transformations \**

**metricName=ErrorCount,metricNamespace=MyNamespace,metricValue=1,defaultValue=0**

You can test this new policy by posting events containing the word "Error" in the message.

To post events to **MyAppErrorCount** using the AWS CLI

At a command prompt, run the following command. Note that patterns are case-sensitive.

**aws logs put-log-events \**

**--log-group-name MyApp/access.log --log-stream-name TestStream1 \**

**--log-events \**

**timestamp=1394793518000,message="This message contains an Error" \**

**timestamp=1394793528000,message="This message also contains an Error"**

### Retention settings

Retention settings can be used to specify how long log events are kept in CloudWatch Logs. Expired log events get deleted automatically. Just like metric filters, retention settings are also assigned to log groups, and the retention assigned to a log group is applied to their log streams.

## Enable Logging on EC2 Instance

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html>

* Start / create a Linux AMI 2 EC2 instance.
* Do some configs:
  + Install CloudWatch Logs Agent.
  + Agent will push logs to CloudWatch.
* Logs can be saved to S3 / Glacier.
* Different steps for different instance types.
* Commands:

$ sudo yum install amazon-cloudwatch-agent

* Verify the signature of the agent package

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/verify-CloudWatch-Agent-Package-Signature.html>

* Create the CloudWatch agent configuration file

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/create-cloudwatch-agent-configuration-file.html>

* Create IAM roles and users for use with CloudWatch agent

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/create-iam-roles-for-cloudwatch-agent-commandline.html>

* + IAM -> Roles -> Create Role.
  + AWS Service.
  + EC2.
  + Next.
  + Permission policies:
    - CloudWatchFullAccess.
    - CloudWatchLogsFullAccess.
  + Next.
  + Role name.
  + Create role.
  + Select EC2 instance -> Actions -> Security -> Modify IAM role.
  + Associate the role created.

# Serverless Computing

*AWS Solution Architect – Module 12-Demo01 with code zip file.*

AWS serverless services help organizations build and deploy native [cloud applications](https://www.koombea.com/services/cloud-application-development/) that utilize a serverless architecture. Serverless applications have become a popular option for companies and development teams because they allow them to focus on the core functionality of their [web](https://www.koombea.com/services/web-development/) or [mobile applications](https://www.koombea.com/services/custom-mobile-app-development/) without spending time maintaining technical infrastructure.

In addition, by using a serverless architecture, organizations can expedite the development process, reduce operational costs, and scale faster. This post will not explain all of the benefits associated with serverless applications. However, it will discuss the most popular AWS serverless services and how they can be used to build powerful serverless applications.

Let’s take a closer look at the most popular AWS services for serverless applications. The most popular AWS services include:

* AWS Lambda
* AWS Fargate
* Amazon SNS
* Amazon SQS
* AWS DynamoDB
* Amazon Aurora
* AWS Step Functions
* Amazon API Gateway

## AWS Lambda

**AWS Lambda is a compute service that enables development teams to run code without provisioning or managing servers**. Developers that use AWS Lambda only have to focus on writing code and not any of the tiresome tasks associated with infrastructure management. In addition, AWS Lambda is an event-driven system. This means that your organization only has to pay for the resources it uses.

Common issues like activity spikes, cold starts, or disaster recovery are no problem for organizations that use AWS Lambda because the cloud environment automatically scales to meet demand. In addition, Lambda can be combined with most other AWS services to create a truly powerful and flexible serverless application.

## AWS Fargate

**AWS Fargate is a serverless compute engine that enables developers to run containers without managing servers or clusters**. Fargate makes deploying, managing and scaling large containerized applications simple. AWS Fargate manages all of the infrastructure required to run your application’s containers and ensures that there is always high availability. With this tool, containerization has never been simpler for development teams.

## Amazon SNS

**Amazon SNS (simple notification service) is a fully managed messaging service**. Amazon SNS can be effectively used for both application-to-application and application-to-user messaging. Developers using Amazon SNS can choose the message type from several popular options, including email, SMS, push notifications, etc.

Amazon SNS allows developers to decouple microservices, serverless applications, and distributed systems. In addition, Amazon SNS can be used for message archiving, ordering, analysis, and filtering.

## Amazon SQS

**Amazon SQS (simple queue service) is a managed, distributed message queueing service**. Like Amazon SNS, SQS enables developers to decouple and scale microservices, serverless applications, and distributed systems. However, SQS does not have a user component. Instead, Amazon SQS sends, receives, and caches messages between software components, microservices, applications, etc.

There are a few other highlights of Amazon SQS that should be covered, such as high-security standards and strong reliability. In addition to the security practices employed by AWS services, Amazon SQS locks messages during processing to provide an extra layer of security. In addition, Amazon SQS stores messages and data on multiple servers to ensure nothing is ever lost.

## AWS DynamoDB

**AWS DynamoDB is a NoSQL document**[**database**](https://www.koombea.com/blog/mongodb-vs-mysql/)**service**. DynamoDB supports key-value pairs and document data structures while delivering single-digit millisecond performance at scale. AWS DynamoDB is a fully-managed service that covers hardware provisioning, setup, configuration, cluster scaling, software patches, and more. In addition, it quickly scales up or down without any dip in performance level.

In addition, AWS DynamoDB meets all compliance requirements and provides data encryption to protect the most sensitive data from falling into the wrong hands. Like AWS Lambda, DynamoDB follows a pay-per-use pricing model, so you only pay for the resources that you actually use. The fact that it can also be combined with a host of other AWS services makes it perfect for serverless applications and cloud app development.

## Amazon Aurora

**Amazon Aurora is a fully-managed relational database engine**. Aurora automatically scales capacity based on your application’s real-time needs. Amazon Aurora stores data in rows, columns, and tables and is fully compatible with MySQL and PostgreSQL. If you already have projects that exist in MySQL or PostgreSQL, you can easily export them into Aurora.

As a fully managed AWS service, Aurora handles time-consuming tasks like hardware provisioning, patching, database setup, storage autoscaling, and backup. AWS also offers an adjacent service called Aurora Serverless, which includes autoscaling services for computing capacity. Aurora Serverless is a great option for organizations with unpredictable and infrequent application workloads.

## AWS Step Functions

**AWS Step Functions gives developers a visual representation of their AWS services and workflows**. Step Functions is a visual workflow orchestrator that enables development teams to sequence their AWS services and track application performance in real time. AWS Step Functions greatly simplifies complex serverless applications with several components and enables organizations to build applications one piece at a time.

## Amazon API Gateway

**Amazon API Gateway is a fully managed service that simplifies API creation and publication at any scale**. APIs rely on consistent communication. AWS is the best platform to ensure that APIs function the way they were intended without fear of failure. This fully managed service takes the mundane, rote responsibilities of API creation and publication out of the hands of your developers and enables them to focus all their efforts on creating great APIs.

## Use case:

In the ‘serverless’ way of doing things, we don’t need to think about servers. Because, the capacity planning, scaling, high availability, and fault tolerance — typically expected of a server — are automatically managed by the Cloud provider. In this case, we will consider AWS (a cloud provider). AWS offers a plethora of services that follow the serverless model, including but not limited to SNS, SQS, Lambda, and SES. One can use these services just by subscribing to them and without actually contacting the server. This eliminates the need to split your attention to ensure all systems are updated and maintained, allowing you to spend more time on the core operations of your business.

Often, companies take feedback from their customers. A book publishing company, for instance, takes the feedback on the newly published book. A cosmetic company takes feedback, on the newly launched cosmetic product. So, after collecting the customer feedback, companies tend to aggregate them all to comprehend the most common issues with the product, and accordingly tweak the product and the marketing strategy designed for the product.

Until now, taking the feedback required creating an EC2 instance, RDS Database, etc. But, in this use case, we will use the `serverless` technologies to build up a feedback website. The services to be used and their purpose are listed below:

* S3 – For storing the web pages
* DynamoDB – For storing the feedback results
* Cognito – To provide the website access to the backend database
* IAM – To specify what permissions to be given



Optional: As an extension to this use case, we can also use AWS QuickSight to perform some visualizations. We can draw a pie chart on how many customers are thinking about the product in a positive/negative way.

Bottom line: Serverless is the way to go, and the cloud service providers are injecting a lot of services/features regularly into their applications. But, as with any other software, it takes some time to attain the maturity stage and win a wider acceptance from the industry.

**AWS Services:** S3, Cognito, IAM, DynamoDB

## Step1: Create a DynamoDB table

1. Region: N. Virginia (us-east)
2. DynamoDB -> Create table
3. Name: survey
4. Partition key: email
5. Sort key: city
6. Create table
7. Navigate to the `Amazon Cognito` Management Console and click on `Create user Pool`.
8. “User pools” in the breadcrumb
9. Federated identities
10. Enter the pool name and check `Enable access to unauthorized identities` and click on `Create Pool` -> Create Pool.
11. Allow on IAM Roles window
12. Change the platform to **JavaScript**, note down the `IdentityPoolId`.



1. In a new browser tab, navigate to IAM -> Roles
2. Filter for the **Cognito** rulesand click on the Role which ends with `PoolUnauth\_Role`.
3. Select Add permissions -> Attach policies
4. Select the `AmazonDynamoDBFullAccess` Policy and click on `Attach policy` or “Add permissions”.
5. Navigate to **S3 Management Console** and create a **Bucket**.
6. Name: ajs-survey-website-01
7. Uncheck “**Block all public access**” -> Acknowledge warning
8. Versioning Disabled
9. Create bucket
10. Select the bucket -> Properties tab
11. Towards the end of the page click on Edit for `Static website hosting`.
12. Select Enable for `Static website hosting`. For the `Index document` enter `survey.html` and for the Error document` enter `error.html`.

**S3 Static Web Site\Survey.html:**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Survey for abc.com</title>

</head>

<body>

<div id="submit-survey">

<b>Your email :</b><input autofocus size="23" type="email" id="email"

placeholder="Enter your email here" /><br /><br />

<b>Your City :</b><input autofocus size="23" type="text" id="city" placeholder="Enter your City here" /><br /><br />

<b>Your feedback :</b><input autofocus size="100" type="text" id="feedback"

placeholder="Enter your feedback here" /><br /><br />

<p id="result"><b>Enter the survey above then click Submit</b></p>

<button class="btn default" onClick="insertSurveyData()">Submit</button>

</div>

<script src="https://sdk.amazonaws.com/js/aws-sdk-2.410.0.min.js"></script>

<script type="text/javascript">

// Initialize the Amazon Cognito credentials provider

AWS.config.region = 'us-east-1';

AWS.config.credentials = new AWS.CognitoIdentityCredentials({ IdentityPoolId: 'us-east-1:4257a04b-b2c4-4a2a-bd39-0e3c938e7114' });

// Function invoked by button click

function insertSurveyData() {

// Create the DynamoDB service object

var ddb = new AWS.DynamoDB({ apiVersion: '2012-08-10' });

var params = {

TableName: 'survey',

Item: {

'email': { S: document.getElementById("email").value },

'city': { S: document.getElementById("city").value },

'feedback': { S: document.getElementById("feedback").value }

}

};

// Call DynamoDB to add the item to the table

ddb.putItem(params, function (err, data) {

if (err) {

document.getElementById('result').innerHTML = error;

} else {

document.getElementById('result').innerHTML = "Thanks for helping with the survey";

}

});

}

</script>

</body>

</html>

**S3 Static Web Site\Error.html:**

<html>

<body>

Oops! Not here.

</body>

</html>

1. Save changes
2. Note down the **URL** at the end under “Static website hosting”. We would be using this to access the web pages in S3 via the browser later.
3. In the Permissions Tab, click on Edit for the `Bucket policy`.
4. Enter the JSON from the attached file into the policy. Make sure to ***replace*** the highlighted S3 bucket name with the bucket name created in one of the previous step (**File: ServerlessComputingPolicy.json**).

{

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

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": [

"s3:GetObject"

],

"Resource": [

"arn:aws:s3:::abc-survey-website/\*"

]

}

]

}

1. Save changes
2. Click on the **Objects tab**. Upload the *survey.html* and the *error.html* to the S3 bucket.
   1. Make sure the `IdentityPoolId` is modified in the survey.html. Use the one got from **the Cognito Console** while creating the **Identity pool.**
   2. Make sure the file names are renamed to survey.html and error.html
3. Use the browser and navigate to the **URL** which was got from the S3 Management Console. Enter the *email, City and feedback*. Click on **Submit**.
4. If everything works fine, then the message `Thanks for helping with the survey` should be displayed.
5. Navigate to the **DynamoDB Management Console** -> **Tables**
6. Click on the **survey** table name -> Explore table items
7. Scroll down to “items returned” and you should see the survey which we have entered in the feedback form should appear in the DynamoDB for further processing.
8. Enter one ore survey on the static site and “Run” this query again on DynamoDB to see the new survey record

# Serverless Lambda Functions with NodeJS

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-nodejs.html>

<https://docs.aws.amazon.com/lambda/latest/dg/nodejs-handler.html>

<https://github.com/awsdocs/aws-lambda-developer-guide/tree/main/sample-apps/nodejs-apig>

Typescript Lambda:

<https://blog.appsignal.com/2022/09/21/how-to-build-aws-lambdas-with-typescript.html>

**Steps**:

Lambda functions use an [execution role](https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html) to get permission to write logs to Amazon CloudWatch Logs, and to access other services and resources. If you don't already have an execution role for function development, create one.

## To create an execution role

1. Open the [roles page](https://console.aws.amazon.com/iam/home#/roles) in the IAM console.
2. Choose **Create role** -> AWS Service -> Lambda -> Next
3. Permissions – AWSLambdaBasicExecutionRole
4. Role name – **lambda-role**.

The **AWSLambdaBasicExecutionRole** policy has the permissions that the function needs to write logs to CloudWatch Logs.

## To create a Node.js function

1. Open the [Lambda console](https://console.aws.amazon.com/lambda).
2. Choose **Create function** -> Author from Scratch
3. Configure the following settings:

* Name – **my-function**.
* Runtime – Node.js 18.x.
* Change default execution role – Use an existing role.
* Existing role – **lambda-role**.

1. Choose Create function.
2. To configure a test event, choose Test.
3. For Event name, enter **test**.
4. Choose Save changes.
5. To invoke the function, choose Test.

The console creates a Lambda function with a single source file named index.js or index.mjs. You can edit this file and add more files in the built-in [code editor](https://docs.aws.amazon.com/lambda/latest/dg/foundation-console.html#code-editor). To save your changes, choose **Save**, then **Deploy**. Then, to run your code, choose **Test**.

## Sample NodeJS Lambda code

**Example 1**:

The following example function logs the contents of the event object and returns the location of the logs.

export const handler = async (event, context) => {

console.log("EVENT: \n" + JSON.stringify(event, null, 2));

return context.logStreamName;

};

**Example 2**:

The next example uses async/await to list your Amazon Simple Storage Service buckets.

**Note**: Before using this example, make sure that your function's execution role has Amazon S3 read permissions.

1. Go to IAM -> Roles -> lambda-role
2. Add Permissions -> Attach Policies
3. Search and select AmazonS3ReadOnlyAccess
4. Select Add Permissions

import {S3Client, ListBucketsCommand} from '@aws-sdk/client-s3';

const s3 = new S3Client({region: 'us-east-1'});

export const handler = async(event) => {

const data = await s3.send(new ListBucketsCommand({}));

return data.Buckets;

};

## Lambda Function -> REST API

<https://dev.to/carlosvldz/rest-api-with-lambda-and-node-js-on-aws-3o2n>

1. Create a new Lambda Function for NodeJS as per previous steps with the following code:

const toLowerCase = async (event) => {

let newSentence = event.sentence.toLowerCase();

const response = {

statusCode: 200,

body: JSON.stringify(newSentence),

};

return response;

};

export const handler = toLowerCase;

1. Test it using the following JSON:

{

"sentence": "HELLO WORLD THIS IS AN EXAMPLE WITH LAMBDA"

}

1. Navigate to API Gateway -> Create API -> REST API (*not REST API private*) -> Build
2. Protocol: REST
3. Create new API: New API
4. API name: simpleAPI
5. Create API
6. Click on the API -> Resources -> Actions -> Create Resource
7. Resource name: lower-case
8. Uncheck CORS
9. Create Resource
10. Select the resource -> Action -> Create Method -> POST (*from dropdown*) -> Save (*tick mark*)
11. Integration type: Lambda function
12. Lambda Function: toLowerCase
13. Save -> OK
14. Actions -> Deploy API
15. Stage: New Stage
16. Stage name: test
17. Deploy
18. Save changes
19. From left pane -> Stages
20. Expand the stage, select the method
21. Copy the “Invoke URL”
22. Open Postman
23. Select POST and enter the URL
24. Select Body -> Raw -> JSON and add this JSON:

{

  "sentence": "HELLO WORLD THIS IS AN EXAMPLE WITH LAMBDA"

}

1. Send
2. Should get result as:

{

    "statusCode": 200,

    "body": "\"hello world this is an example with lambda\""

}

## Using Lambda with API Gateway - NodeJS

<https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway-tutorial.html>

In this tutorial, you create a REST API through which you invoke a Lambda function using an HTTP request. Your Lambda function will perform create, read, update, and delete (CRUD) operations on a DynamoDB table. This function is provided here for demonstration, but you will learn to configure an API Gateway REST API that can invoke any Lambda function.


      Services and resources used in this tutorial
    

Using API Gateway provides users with a secure HTTP endpoint to invoke your Lambda function and can help manage large volumes of calls to your function by throttling traffic and automatically validating and authorizing API calls. API Gateway also provides flexible security controls using AWS Identity and Access Management (IAM) and Amazon Cognito. This is useful for use cases where advance authorization is required for calls to your application.

To complete this tutorial, you will go through the following stages:

1. Create and configure a Lambda function in Python or Node.js to perform operations on a DynamoDB table.
2. Create a REST API in API Gateway to connect to your Lambda function.
3. Create a DynamoDB table and test it with your Lambda function in the console.
4. Deploy your API and test the full setup using curl in a terminal.

By completing these stages, you will learn how to use API Gateway to create an HTTP endpoint that can securely invoke a Lambda function at any scale. You will also learn how to deploy your API, and how to test it in the console and by sending an HTTP request using a terminal.

### Prerequisites

#### Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

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

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

When you sign up for an AWS account, an AWS account root user is created. The root user has access to all AWS services and resources in the account. As a security best practice, [assign administrative access to an administrative user](https://docs.aws.amazon.com/singlesignon/latest/userguide/getting-started.html), and use only the root user to perform [tasks that require root user access](https://docs.aws.amazon.com/accounts/latest/reference/root-user-tasks.html).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

#### Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

**Secure your AWS account root user**

1. Sign in to the [AWS Management Console](https://console.aws.amazon.com/) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.
2. For help signing in by using root user, see [Signing in as the root user](https://docs.aws.amazon.com/signin/latest/userguide/console-sign-in-tutorials.html#introduction-to-root-user-sign-in-tutorial) in the AWS Sign-In User Guide.
3. Turn on multi-factor authentication (MFA) for your root user.
4. For instructions, see [Enable a virtual MFA device for your AWS account root user (console)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_virtual.html#enable-virt-mfa-for-root) in the IAM User Guide.

**Create an administrative user**

* For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center (successor to AWS Single Sign-On).

**Sign in as the administrative user**

* To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

#### Install the AWS Command Line Interface

If you have not yet installed the AWS Command Line Interface, follow the steps at [Installing or updating the latest version of the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html) to install it.

The tutorial requires a command line terminal or shell to run commands. In Linux and macOS, use your preferred shell and package manager.

### Create a permissions policy


        Step 1 create a permissions policy
      

Before you can create an [execution role](https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html) for you Lambda function, you first need to create a permissions policy to give your function permission to access the required AWS resources. For this tutorial, the policy allows Lambda to perform CRUD operations on a DynamoDB table and write to Amazon CloudWatch Logs.

To create the policy

1. Open the [Policies page](https://console.aws.amazon.com/iam/home#/policies) of the IAM console.
2. Choose **Create Policy**.
3. Choose the **JSON** tab, and then paste the following custom policy into the JSON editor.

{

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

"Statement": [

{

"Sid": "Stmt1428341300017",

"Action": [

"dynamodb:DeleteItem",

"dynamodb:GetItem",

"dynamodb:PutItem",

"dynamodb:Query",

"dynamodb:Scan",

"dynamodb:UpdateItem"

],

"Effect": "Allow",

"Resource": "\*"

},

{

"Sid": "",

"Resource": "\*",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Effect": "Allow"

}

]

}

1. Choose Next: Tags.
2. Choose Next: Review.
3. Under Review policy, for the policy Name, enter **lambda-apigateway-policy**.
4. Choose Create policy.

### Create an execution role


        Step 2 create an execution role
      

An execution role is an AWS Identity and Access Management (IAM) role that grants a Lambda function permission to access AWS services and resources. To enable your function to perform operations on a DynamoDB table, you attach the permissions policy you created in the previous step.

To create an execution role and attach your custom permissions policy

1. Open the [Roles page](https://console.aws.amazon.com/iam/home#/roles) of the IAM console.
2. Choose **Create role**.
3. For the type of trusted entity, choose **AWS service**, then for the use case, choose **Lambda**.
4. Choose **Next**.
5. In the policy search box, enter **lambda-apigateway-policy**.
6. In the search results, select the policy that you created (lambda-apigateway-policy), and then choose **Next**.
7. Under **Role details**, for the **Role name**, enter **lambda-apigateway-role**, then choose **Create role**.

Later in the tutorial, you need the Amazon Resource Name (ARN) of the role you just created. On the **Roles** page of the IAM console, choose the name of your role (lambda-apigateway-role) and copy the **Role ARN** displayed on the **Summary** page.

### Create the function


        Step 3 create the function
      

The following code example receives an event input from API Gateway specifying an operation to perform on the DynamoDB table you will create and some payload data. If the parameters the function receives are valid, it performs the requested operation on the table.

**Example index.mjs**

console.log('Loading function');

import { DynamoDBDocumentClient, PutCommand, GetCommand,

UpdateCommand, DeleteCommand} from "@aws-sdk/lib-dynamodb";

import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const ddbClient = new DynamoDBClient({ region: "us-east-1" });

const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);

*// Define the name of the DDB table to perform the CRUD operations on*

const tablename = "lambda-apigateway";

*/\*\**

*\* Provide an event that contains the following keys:*

*\**

*\* - operation: one of 'create,' 'read,' 'update,' 'delete,' or 'echo'*

*\* - payload: a JSON object containing the parameters for the table item*

*\* to perform the operation on*

*\*/*

export const handler = async (event, context) => {

const operation = event.operation;

if (operation == 'echo'){

return(event.payload);

}

else {

event.payload.TableName = tablename;

switch (operation) {

case 'create':

await ddbDocClient.send(new PutCommand(event.payload));

break;

case 'read':

var table\_item = await ddbDocClient.send(new GetCommand(event.payload));

console.log(table\_item);

break;

case 'update':

await ddbDocClient.send(new UpdateCommand(event.payload));

break;

case 'delete':

await ddbDocClient.send(new DeleteCommand(event.payload));

break;

default:

return ('Unknown operation: ${operation}');

}

}

};

**Note**: In this example, the name of the DynamoDB table is defined as a variable in your function code. In a real application, best practice is to pass this parameter as an environment variable and to avoid hardcoding the table name.

To create the function

1. Save the code example as a file named index.mjs and, if necessary, edit the AWS region specified in the code. The region specified in the code must be the same as the region in which you create your DynamoDB table later in the tutorial.
2. Create a deployment package using the following zip command.

**zip function.zip index.mjs**

1. Create a Lambda function using the create-function AWS CLI command. For the role parameter, enter the execution role's Amazon Resource Name (ARN) that you copied earlier.

**aws lambda create-function --function-name LambdaFunctionOverHttps \**

**--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \**

**--role *arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role***

### Invoke the function using the AWS CLI


        Step 4 invoke the function using the AWS CLI
      

Before integrating your function with API Gateway, confirm that you have deployed the function successfully. Create a test event containing the parameters your API Gateway API will send to Lambda and use the AWS CLI invoke command to run your function.

To invoke the Lambda function with the AWS CLI

1. Save the following JSON as a file named input.txt.

{

"operation": "echo",

"payload": {

"somekey1": "somevalue1",

"somekey2": "somevalue2"

}

}

1. Run the following invoke AWS CLI command.

**aws lambda invoke --function-name LambdaFunctionOverHttps \**

**--payload file://input.txt outputfile.txt \**

**--cli-binary-format raw-in-base64-out**

The **cli-binary-format** option is required if you're using AWS CLI version 2. To make this the default setting, run aws configure set cli-binary-format raw-in-base64-out. For more information, see [AWS CLI supported global command line options](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-options.html#cli-configure-options-list).

You should see the following response:

{

"StatusCode": 200,

"ExecutedVersion": "LATEST"

}

1. Confirm that your function performed the echo operation you specified in the JSON test event. Inspect the outputfile.txt file and verify it contains the following:

{"somekey1": "somevalue1", "somekey2": "somevalue2"}

### Create a REST API using API Gateway


        Step 5 create the API
      

In this step, you create the API Gateway REST API you will use to invoke your Lambda function.

To create the API

1. Open the [API Gateway console](https://console.aws.amazon.com/apigateway).
2. Choose **Create API**.
3. In the **REST API** box, choose **Build**.
4. Under **Settings**, for **API Name** enter **DynamoDBOperations**.
5. Choose **Create API**.

### Create a resource on your REST API

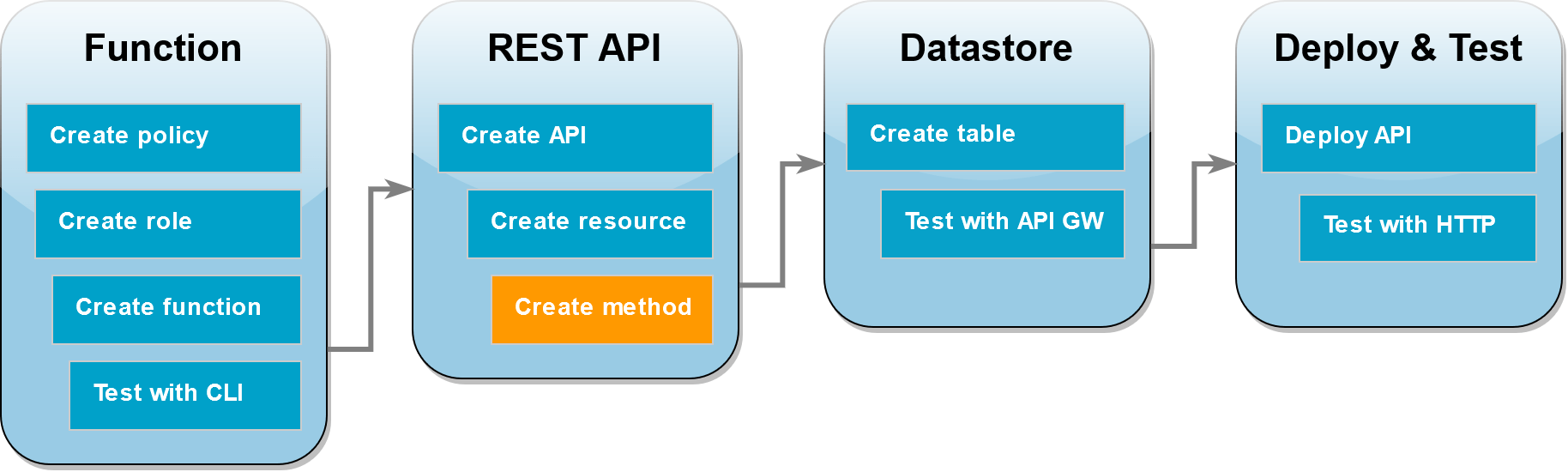

        Step 6 create the API resource
      

To add an HTTP method to your API, you first need to create a resource for that method to operate on. Here you create the resource to manage your DynamoDB table.

To create the resource

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), in the **Resources** tree of your API, make sure that the root (/) level is highlighted. Then, choose **Actions**, **Create Resource**.
2. Under **New child resource**, do the following:
   1. For **Resource Name**, enter **DynamoDBManager**.
   2. Keep **Resource Path** set to /dynamodbmanager.
3. Choose **Create Resource**.

### Create an HTTP POST method



In this step, you create a method (POST) for your DynamoDBManager resource. You link this POST method to your Lambda function so that when the method receives an HTTP request, API Gateway invokes your Lambda function.

**Note**: For the purpose of this tutorial, one HTTP method (POST) is used to invoke a single Lambda function which carries out all of the operations on your DynamoDB table. In a real application, best practice is to use a different Lambda function and HTTP method for each operation. For more information, see [The Lambda monolith](https://docs.aws.amazon.com/lambda/latest/operatorguide/monolith.html).

To create the POST method

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), in the **Resources** tree of your API, make sure that /dynamodbmanager is highlighted. Then, choose **Actions**, **Create Method**.
2. In the small dropdown menu that appears under /dynamodbmanager, choose POST, and then choose the check mark icon.
3. In the method's **Setup** pane, do the following:
   1. For **Integration type**, choose **Lambda Function**.
   2. For **Lambda Region**, choose the same AWS Region as your Lambda function.
   3. For **Lambda Function**, enter the name of your function (**LambdaFunctionOverHttps**).
   4. Select **Use Default Timeout**.
   5. Choose **Save**.
4. In the **Add Permission to Lambda Function** dialog box, choose **OK**.

### Create a DynamoDB table


        Step 8 create a DynamoDB table
      

Create an empty DynamoDB table that your Lambda function will perform CRUD operations on.

To create the DynamoDB table

1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console.
2. Choose **Create table**.
3. Under **Table details**, do the following:
   1. For **Table name**, enter **lambda-apigateway**.
   2. For **Partition key**, enter **id**, and keep the data type set as **String**.
4. Under **Table settings**, keep the **Default settings**.
5. Choose **Create table**.

### Test the integration of API Gateway, Lambda, and DynamoDB


        Step 9 test the integration of API Gateway, Lambda, and DynamoDB
      

You're now ready to test the integration of your API Gateway API method with your Lambda function and your DynamoDB table. Using the API Gateway console, you send requests directly to your POST method using the console's test function. In this step, you first use a create operation to add a new item to your DynamoDB table, then you use an update operation to modify the item.

#### Test 1: To create a new item in your DynamoDB table

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), choose your API (DynamoDBOperations).
2. In the **Resources** tree, under /dynamodbmanager, choose your POST method.
3. In the **Method Execution** pane, in the **Client** box, choose **Test**.
4. In the **Method Test** pane, keep **Query Strings** and **Headers** empty. For **Request Body**, paste the following JSON:

{

"operation": "create",

"payload": {

"Item": {

"id": "1234ABCD",

"number": 5

}

}

}

1. Choose **Test**.

The results that are displayed when the test completes should show status 200. This status code indicates that the create operation was successful.

To confirm, check that your DynamoDB table now contains the new item.

1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console and choose the lambda-apigateway table.
2. Chose **Explore table items**. In the **Items returned** pane, you should see one item with the **id** 1234ABCD and the **number** 5.

#### Test 2: To update the item in your DynamoDB table

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), return to your POST method's **Method Test** pane.
2. In the **Method Test** pane, keep **Query Strings** and **Headers** empty. In **Request Body**, paste the following JSON:

{

"operation": "update",

"payload": {

"Key": {

"id": "1234ABCD"

},

"AttributeUpdates": {

"number": {

"Value": 10

}

}

}

}

1. Choose **Test**.

The results which are displayed when the test completes should show status 200. This status code indicates that the update operation was successful.

To confirm, check that the item in yout DynamoDB table has been modified.

1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console and choose the lambda-apigateway table.
2. Chose **Explore table items**. In the **Items returned** pane, you should see one item with the **id** 1234ABCD and the **number** 10.

### Deploy the API

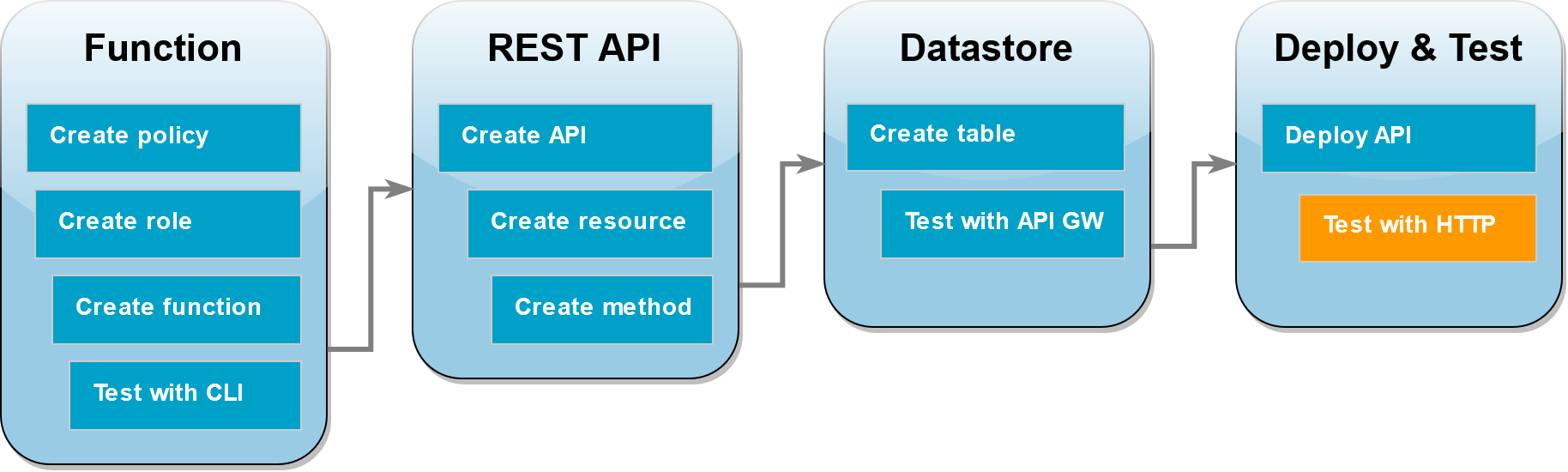

        Step 10 deploy the API
      

For a client to call the API, you must create a deployment and an associated stage. A stage represents a snapshot of your API including its methods and integrations.

To deploy the API

1. Open the **APIs** page of the [API Gateway console](https://console.aws.amazon.com/apigateway) and choose the DynamoDBOperations API.
2. Choose **Actions**, **Deploy API**.
3. For **Deployment stage**, choose **[New Stage]**, then for **Stage name**, enter **test**.
4. Choose **Deploy**.
5. In the **test Stage Editor** pane, copy the **Invoke URL**. You will use this in the next step to invoke your function using an HTTP request.

### Use curl to invoke your function using HTTP requests



You can now invoke your Lambda function by issuing an HTTP request to your API. In this step, you will create a new item in your DynamoDB table and then delete it.

To invoke the Lambda function using curl

1. Run the following curl command using the invoke URL you copied in the previous step. When you use curl with the -d (data) option, it automatically uses the HTTP POST method.

**curl** [**https://*l8togsqxd8.execute-api.us-west-2.amazonaws.com/test*/dynamodbmanager \**](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager%20\)

**-d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'**

1. To verify that the create operation was successful, do the following:
   1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console and choose the lambda-apigateway table.
   2. Choose **Explore table items**. In the **Items returned** pane, you should see an item with the **id** 5678EFGH and the **number** 15.
2. Run the following **curl** command to delete the item you just created. Use your own invoke URL.

**curl** [**https://*l8togsqxd8.execute-api.us-west-2.amazonaws.com/test*/dynamodbmanager \**](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager%20\)

**-d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'**

1. Confirm that the delete operation was successful. In the **Items returned** pane of the DynamoDB console **Explore items** page, verify that the item with **id** 5678EFGH is no longer in the table.

### Clean up your resources (optional)

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

To delete the Lambda function

1. Open the [Functions page](https://console.aws.amazon.com/lambda/home#/functions) of the Lambda console.
2. Select the function that you created.
3. Choose **Actions**, **Delete**.
4. Type **delete** in the text input field and choose **Delete**.

To delete the execution role

1. Open the [Roles page](https://console.aws.amazon.com/iam/home#/roles) of the IAM console.
2. Select the execution role that you created.
3. Choose **Delete**.
4. Enter the name of the role in the text input field and choose **Delete**.

To delete the policy

1. Open the [Policies page](https://console.aws.amazon.com/iam/home#/policies) of the IAM console.
2. Select the policy that you created.
3. Choose **Delete**.
4. Enter the name of the policy in the text input field and choose **Delete**.

To delete the API

1. Open the [APIs page](https://console.aws.amazon.com/apigateway/main/apis) of the API Gateway console.
2. Select the API you created.
3. Choose **Actions**, **Delete**.
4. Choose **Delete**.

To delete the DynamoDB table

1. Open the [Tables page](https://console.aws.amazon.com/dynamodb/home#tables:) of the DynamoDB console.
2. Select the table you created.
3. Choose **Delete**.
4. Enter **delete** in the text box.
5. Choose **Delete table**.

# Serverless Lambda Functions with Python

<https://stackify.com/aws-lambda-with-python-a-complete-getting-started-guide/>

<https://adamtheautomator.com/aws-lambda-python/>

<https://medium.com/@stephinmon.antony/aws-lambda-with-python-examples-2eb227f5fafe>

## Using Lambda with API Gateway - Python

<https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway-tutorial.html>

In this tutorial, you create a REST API through which you invoke a Lambda function using an HTTP request. Your Lambda function will perform create, read, update, and delete (CRUD) operations on a DynamoDB table. This function is provided here for demonstration, but you will learn to configure an API Gateway REST API that can invoke any Lambda function.


      Services and resources used in this tutorial
    

Using API Gateway provides users with a secure HTTP endpoint to invoke your Lambda function and can help manage large volumes of calls to your function by throttling traffic and automatically validating and authorizing API calls. API Gateway also provides flexible security controls using AWS Identity and Access Management (IAM) and Amazon Cognito. This is useful for use cases where advance authorization is required for calls to your application.

To complete this tutorial, you will go through the following stages:

1. Create and configure a Lambda function in Python or Node.js to perform operations on a DynamoDB table.
2. Create a REST API in API Gateway to connect to your Lambda function.
3. Create a DynamoDB table and test it with your Lambda function in the console.
4. Deploy your API and test the full setup using curl in a terminal.

By completing these stages, you will learn how to use API Gateway to create an HTTP endpoint that can securely invoke a Lambda function at any scale. You will also learn how to deploy your API, and how to test it in the console and by sending an HTTP request using a terminal.

### Prerequisites

#### Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

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

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

When you sign up for an AWS account, an AWS account root user is created. The root user has access to all AWS services and resources in the account. As a security best practice, [assign administrative access to an administrative user](https://docs.aws.amazon.com/singlesignon/latest/userguide/getting-started.html), and use only the root user to perform [tasks that require root user access](https://docs.aws.amazon.com/accounts/latest/reference/root-user-tasks.html).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

#### Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

**Secure your AWS account root user**

1. Sign in to the [AWS Management Console](https://console.aws.amazon.com/) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.
2. For help signing in by using root user, see [Signing in as the root user](https://docs.aws.amazon.com/signin/latest/userguide/console-sign-in-tutorials.html#introduction-to-root-user-sign-in-tutorial) in the AWS Sign-In User Guide.
3. Turn on multi-factor authentication (MFA) for your root user.
4. For instructions, see [Enable a virtual MFA device for your AWS account root user (console)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_virtual.html#enable-virt-mfa-for-root) in the IAM User Guide.

**Create an administrative user**

* For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center (successor to AWS Single Sign-On).

**Sign in as the administrative user**

* To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

#### Install the AWS Command Line Interface

If you have not yet installed the AWS Command Line Interface, follow the steps at [Installing or updating the latest version of the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html) to install it.

The tutorial requires a command line terminal or shell to run commands. In Linux and macOS, use your preferred shell and package manager.

### Create a permissions policy


        Step 1 create a permissions policy
      

Before you can create an [execution role](https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html) for you Lambda function, you first need to create a permissions policy to give your function permission to access the required AWS resources. For this tutorial, the policy allows Lambda to perform CRUD operations on a DynamoDB table and write to Amazon CloudWatch Logs.

To create the policy

1. Open the [Policies page](https://console.aws.amazon.com/iam/home#/policies) of the IAM console.
2. Choose **Create Policy**.
3. Choose the **JSON** tab, and then paste the following custom policy into the JSON editor.

{

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

"Statement": [

{

"Sid": "Stmt1428341300017",

"Action": [

"dynamodb:DeleteItem",

"dynamodb:GetItem",

"dynamodb:PutItem",

"dynamodb:Query",

"dynamodb:Scan",

"dynamodb:UpdateItem"

],

"Effect": "Allow",

"Resource": "\*"

},

{

"Sid": "",

"Resource": "\*",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Effect": "Allow"

}

]

}

1. Choose Next: Tags.
2. Choose Next: Review.
3. Under Review policy, for the policy Name, enter **lambda-apigateway-policy**.
4. Choose Create policy.

### Create an execution role


        Step 2 create an execution role
      

An execution role is an AWS Identity and Access Management (IAM) role that grants a Lambda function permission to access AWS services and resources. To enable your function to perform operations on a DynamoDB table, you attach the permissions policy you created in the previous step.

To create an execution role and attach your custom permissions policy

1. Open the [Roles page](https://console.aws.amazon.com/iam/home#/roles) of the IAM console.
2. Choose **Create role**.
3. For the type of trusted entity, choose **AWS service**, then for the use case, choose **Lambda**.
4. Choose **Next**.
5. In the policy search box, enter **lambda-apigateway-policy**.
6. In the search results, select the policy that you created (lambda-apigateway-policy), and then choose **Next**.
7. Under **Role details**, for the **Role name**, enter **lambda-apigateway-role**, then choose **Create role**.

Later in the tutorial, you need the Amazon Resource Name (ARN) of the role you just created. On the **Roles** page of the IAM console, choose the name of your role (lambda-apigateway-role) and copy the **Role ARN** displayed on the **Summary** page.

### Create the function


        Step 3 create the function
      

The following code example receives an event input from API Gateway specifying an operation to perform on the DynamoDB table you will create and some payload data. If the parameters the function receives are valid, it performs the requested operation on the table.

**Example LambdaFunctionOverHttps.py**

import boto3

import json

*# define the DynamoDB table that Lambda will connect to*

tableName = "lambda-apigateway"

*# create the DynamoDB resource*

dynamo = boto3.resource('dynamodb').Table(tableName)

print('Loading function')

def handler(event, context):

'''Provide an event that contains the following keys:

- operation: one of the operations in the operations dict below

- payload: a JSON object containing parameters to pass to the

operation being performed

'''

*# define the functions used to perform the CRUD operations*

def ddb\_create(x):

dynamo.put\_item(\*\*x)

def ddb\_read(x):

dynamo.get\_item(\*\*x)

def ddb\_update(x):

dynamo.update\_item(\*\*x)

def ddb\_delete(x):

dynamo.delete\_item(\*\*x)

def echo(x):

return x

operation = event['operation']

operations = {

'create': ddb\_create,

'read': ddb\_read,

'update': ddb\_update,

'delete': ddb\_delete,

'echo': echo,

}

if operation in operations:

return operations[operation](event.get('payload'))

else:

raise ValueError('Unrecognized operation "{}"'.format(operation))

**Note**: In this example, the name of the DynamoDB table is defined as a variable in your function code. In a real application, best practice is to pass this parameter as an environment variable and to avoid hardcoding the table name.

To create the function

1. Save the code example as a file named LambdaFunctionOverHttps.py.
2. Create a deployment package using the following zip command.

**zip function.zip LambdaFunctionOverHttps.py**

1. Create a Lambda function using the create-function AWS CLI command. For the role parameter, enter the execution role's Amazon Resource Name (ARN) that you copied earlier.

**aws lambda create-function --function-name LambdaFunctionOverHttps \**

**--zip-file fileb://function.zip –-handler \**

**LambdaFunctionOverHttps.handler --runtime python3.9 \**

**--role *arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role***

### Invoke the function using the AWS CLI


        Step 4 invoke the function using the AWS CLI
      

Before integrating your function with API Gateway, confirm that you have deployed the function successfully. Create a test event containing the parameters your API Gateway API will send to Lambda and use the AWS CLI invoke command to run your function.

To invoke the Lambda function with the AWS CLI

1. Save the following JSON as a file named input.txt.

{

"operation": "echo",

"payload": {

"somekey1": "somevalue1",

"somekey2": "somevalue2"

}

}

1. Run the following invoke AWS CLI command.

**aws lambda invoke --function-name LambdaFunctionOverHttps \**

**--payload file://input.txt outputfile.txt \**

**--cli-binary-format raw-in-base64-out**

The **cli-binary-format** option is required if you're using AWS CLI version 2. To make this the default setting, run aws configure set cli-binary-format raw-in-base64-out. For more information, see [AWS CLI supported global command line options](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-options.html#cli-configure-options-list).

You should see the following response:

{

"StatusCode": 200,

"ExecutedVersion": "LATEST"

}

1. Confirm that your function performed the echo operation you specified in the JSON test event. Inspect the outputfile.txt file and verify it contains the following:

{"somekey1": "somevalue1", "somekey2": "somevalue2"}

### Create a REST API using API Gateway


        Step 5 create the API
      

In this step, you create the API Gateway REST API you will use to invoke your Lambda function.

To create the API

1. Open the [API Gateway console](https://console.aws.amazon.com/apigateway).
2. Choose **Create API**.
3. In the **REST API** box, choose **Build**.
4. Under **Settings**, for **API Name** enter **DynamoDBOperations**.
5. Choose **Create API**.

### Create a resource on your REST API

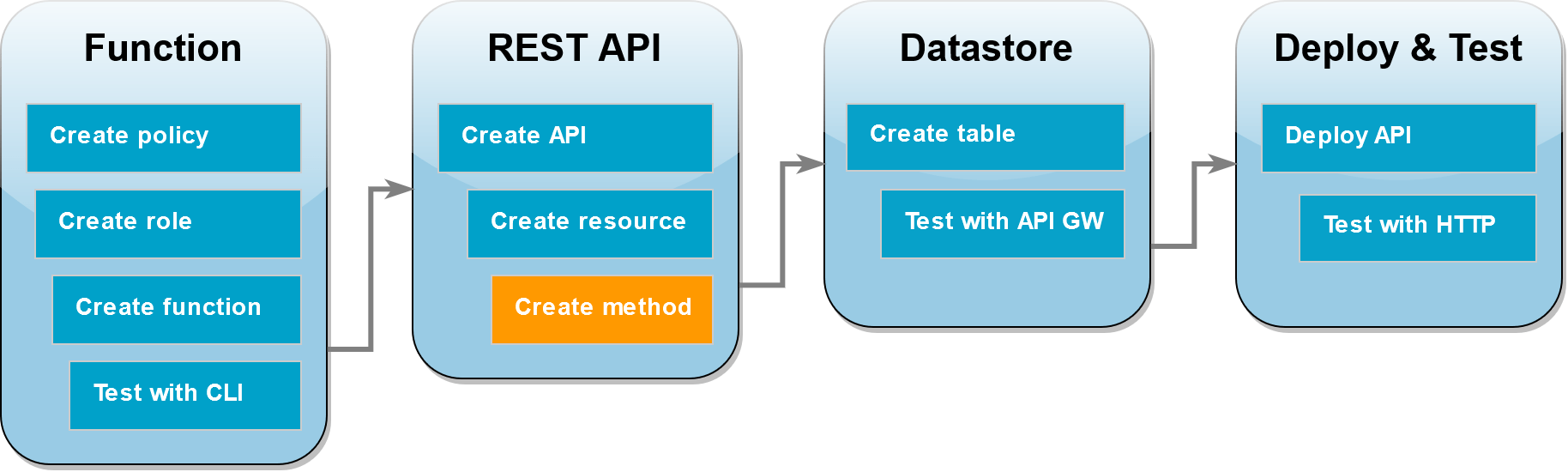

        Step 6 create the API resource
      

To add an HTTP method to your API, you first need to create a resource for that method to operate on. Here you create the resource to manage your DynamoDB table.

To create the resource

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), in the **Resources** tree of your API, make sure that the root (/) level is highlighted. Then, choose **Actions**, **Create Resource**.
2. Under **New child resource**, do the following:
   1. For **Resource Name**, enter **DynamoDBManager**.
   2. Keep **Resource Path** set to /dynamodbmanager.
3. Choose **Create Resource**.

### Create an HTTP POST method



In this step, you create a method (POST) for your DynamoDBManager resource. You link this POST method to your Lambda function so that when the method receives an HTTP request, API Gateway invokes your Lambda function.

**Note**: For the purpose of this tutorial, one HTTP method (POST) is used to invoke a single Lambda function which carries out all of the operations on your DynamoDB table. In a real application, best practice is to use a different Lambda function and HTTP method for each operation. For more information, see [The Lambda monolith](https://docs.aws.amazon.com/lambda/latest/operatorguide/monolith.html).

To create the POST method

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), in the **Resources** tree of your API, make sure that /dynamodbmanager is highlighted. Then, choose **Actions**, **Create Method**.
2. In the small dropdown menu that appears under /dynamodbmanager, choose POST, and then choose the check mark icon.
3. In the method's **Setup** pane, do the following:
   1. For **Integration type**, choose **Lambda Function**.
   2. For **Lambda Region**, choose the same AWS Region as your Lambda function.
   3. For **Lambda Function**, enter the name of your function (**LambdaFunctionOverHttps**).
   4. Select **Use Default Timeout**.
   5. Choose **Save**.
4. In the **Add Permission to Lambda Function** dialog box, choose **OK**.

### Create a DynamoDB table


        Step 8 create a DynamoDB table
      

Create an empty DynamoDB table that your Lambda function will perform CRUD operations on.

To create the DynamoDB table

1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console.
2. Choose **Create table**.
3. Under **Table details**, do the following:
   1. For **Table name**, enter **lambda-apigateway**.
   2. For **Partition key**, enter **id**, and keep the data type set as **String**.
4. Under **Table settings**, keep the **Default settings**.
5. Choose **Create table**.

### Test the integration of API Gateway, Lambda, and DynamoDB


        Step 9 test the integration of API Gateway, Lambda, and DynamoDB
      

You're now ready to test the integration of your API Gateway API method with your Lambda function and your DynamoDB table. Using the API Gateway console, you send requests directly to your POST method using the console's test function. In this step, you first use a create operation to add a new item to your DynamoDB table, then you use an update operation to modify the item.

#### Test 1: To create a new item in your DynamoDB table

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), choose your API (DynamoDBOperations).
2. In the **Resources** tree, under /dynamodbmanager, choose your POST method.
3. In the **Method Execution** pane, in the **Client** box, choose **Test**.
4. In the **Method Test** pane, keep **Query Strings** and **Headers** empty. For **Request Body**, paste the following JSON (add-item.json):

{

"operation": "create",

"payload": {

"Item": {

"id": "1234ABCD",

"number": 5

}

}

}

1. Choose **Test**.

The results that are displayed when the test completes should show status 200. This status code indicates that the create operation was successful.

To confirm, check that your DynamoDB table now contains the new item.

1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console and choose the lambda-apigateway table.
2. Chose **Explore table items**. In the **Items returned** pane, you should see one item with the **id** 1234ABCD and the **number** 5.

#### Test 2: To update the item in your DynamoDB table

1. In the [API Gateway console](https://console.aws.amazon.com/apigateway), return to your POST method's **Method Test** pane.
2. In the **Method Test** pane, keep **Query Strings** and **Headers** empty. In **Request Body**, paste the following JSON (update-item.json):

{

"operation": "update",

"payload": {

"Key": {

"id": "1234ABCD"

},

"AttributeUpdates": {

"number": {

"Value": 10

}

}

}

}

1. Choose **Test**.

The results which are displayed when the test completes should show status 200. This status code indicates that the update operation was successful.

To confirm, check that the item in yout DynamoDB table has been modified.

1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console and choose the lambda-apigateway table.
2. Chose **Explore table items**. In the **Items returned** pane, you should see one item with the **id** 1234ABCD and the **number** 10.

### Deploy the API

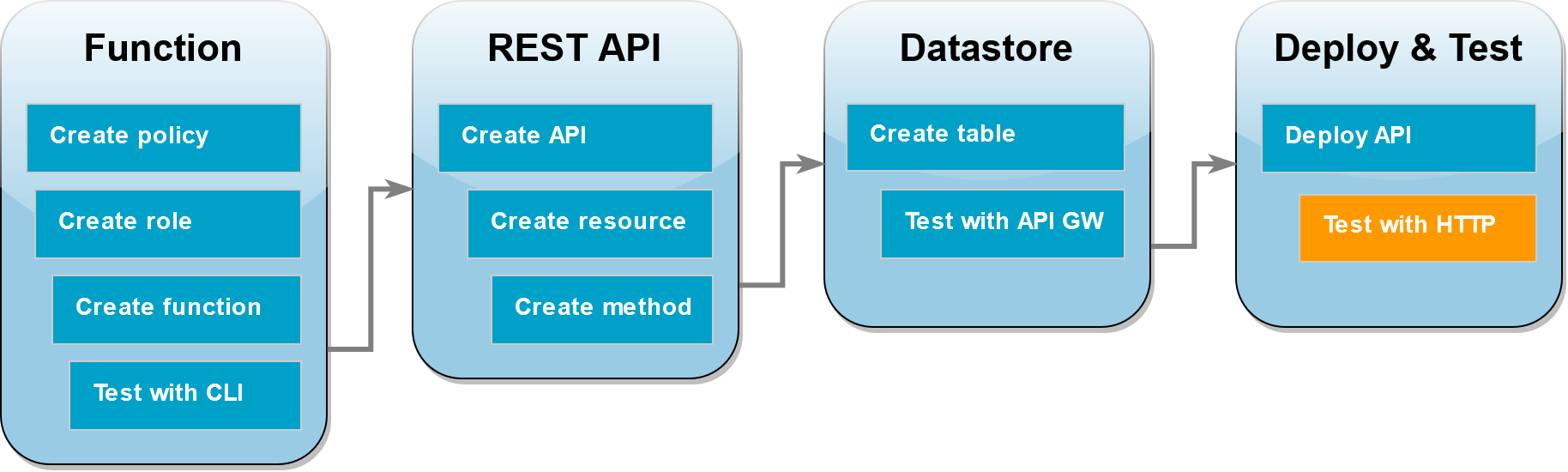

        Step 10 deploy the API
      

For a client to call the API, you must create a deployment and an associated stage. A stage represents a snapshot of your API including its methods and integrations.

To deploy the API

1. Open the **APIs** page of the [API Gateway console](https://console.aws.amazon.com/apigateway) and choose the DynamoDBOperations API.
2. Choose **Actions**, **Deploy API**.
3. For **Deployment stage**, choose **[New Stage]**, then for **Stage name**, enter **test**.
4. Choose **Deploy**.
5. In the **test Stage Editor** pane, copy the **Invoke URL**. You will use this in the next step to invoke your function using an HTTP request.

### Use curl to invoke your function using HTTP requests



You can now invoke your Lambda function by issuing an HTTP request to your API. In this step, you will create a new item in your DynamoDB table and then delete it.

To invoke the Lambda function using curl

1. Run the following curl command using the invoke URL you copied in the previous step. When you use curl with the -d (data) option, it automatically uses the HTTP POST method.

**On Windows:**

**curl** [**https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager**](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager) **-d "{\"operation\": \"create\", \"payload\": {\"Item\": {\"id\": \"5678EFGH\", \"number\": 15}}}"**

**On Linux**:

**curl [https://](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager \\)*[l8togsqxd8.execute-api.us-west-2.amazonaws.com/test](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager \\)*[/dynamodbmanager \](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager \\)**

**-d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'**

1. To verify that the create operation was successful, do the following:
   1. Open the [Tables page](https://console.aws.amazon.com/dynamodbv2#tables) of the DynamoDB console and choose the lambda-apigateway table.
   2. Choose **Explore table items**. In the **Items returned** pane, you should see an item with the **id** 5678EFGH and the **number** 15.
2. Run the following **curl** command to delete the item you just created. Use your own invoke URL.

**On Windows:**

**curl** [**https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager**](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager) **-d "{\"operation\": \"delete\", \"payload\": {\"Key\": {\"id\": \"5678EFGH\"}}}"**

**On linux:**

**curl** [**https://*l8togsqxd8.execute-api.us-west-2.amazonaws.com/test*/dynamodbmanager \**](https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/dynamodbmanager%20\)

**-d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'**

1. Confirm that the delete operation was successful. In the **Items returned** pane of the DynamoDB console **Explore items** page, verify that the item with **id** 5678EFGH is no longer in the table.

### Clean up your resources (optional)

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

To delete the Lambda function

1. Open the [Functions page](https://console.aws.amazon.com/lambda/home#/functions) of the Lambda console.
2. Select the function that you created.
3. Choose **Actions**, **Delete**.
4. Type **delete** in the text input field and choose **Delete**.

To delete the execution role

1. Open the [Roles page](https://console.aws.amazon.com/iam/home#/roles) of the IAM console.
2. Select the execution role that you created.
3. Choose **Delete**.
4. Enter the name of the role in the text input field and choose **Delete**.

To delete the policy

1. Open the [Policies page](https://console.aws.amazon.com/iam/home#/policies) of the IAM console.
2. Select the policy that you created.
3. Choose **Delete**.
4. Enter the name of the policy in the text input field and choose **Delete**.

To delete the API

1. Open the [APIs page](https://console.aws.amazon.com/apigateway/main/apis) of the API Gateway console.
2. Select the API you created.
3. Choose **Actions**, **Delete**.
4. Choose **Delete**.

To delete the DynamoDB table

1. Open the [Tables page](https://console.aws.amazon.com/dynamodb/home#tables:) of the DynamoDB console.
2. Select the table you created.
3. Choose **Delete**.
4. Enter **delete** in the text box.
5. Choose **Delete table**.

# AWS CLI and SDK

*AWS Developer Certification training -> Module 4 – Recording and Presentation*

## AWS CLI Installation

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

1. From command prompt, run:

C:\> **msiexec.exe /i** [**https://awscli.amazonaws.com/AWSCLIV2.msi**](https://awscli.amazonaws.com/AWSCLIV2.msi)

1. Installed in C:\Program Files\Amazon\AWSCLIV2\
2. To confirm the installation, open the **Start** menu, search for cmd to open a command prompt window, and at the command prompt use the aws --version command.

C:\> **aws --version**

1. Make sure you have created a non-root user on AWS with **AdministratorAccess** permissions.
2. Configure user with AWS CLI

C:\> **aws configure**

1. Enter the following for the user:

AWS Access Key ID [None]: ***AKIAIOSFODNN7EXAMPLE***

AWS Secret Access Key [None]: ***wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY***

Default region name [None]: ***us-east-1***

Default output format [None]: ***json***

1. The AWS CLI stores sensitive credential information that you specify with aws configure in a local file named credentials, in a folder named .aws in your home directory. The less sensitive configuration options that you specify with aws configure are stored in a local file named config, also stored in the .aws folder in your home directory.
   1. Where you find your home directory location varies based on the operating system, but is referred to using the environment variables %UserProfile% in Windows and $HOME or ~ (tilde) in Unix-based systems.

**Using Named Profiles**:

If no profile is explicitly defined, the default profile is used.

To use a named profile, add the --profile *profile-name* option to your command. The following example lists all of your Amazon EC2 instances using the credentials and settings defined in the user1 profile.

**aws ec2 describe-instances --profile user1**

To use a named profile for multiple commands, you can avoid specifying the profile in every command by setting the AWS\_PROFILE environment variable as the default profile. You can override this setting by using the --profile parameter.

Linux:

$ **export AWS\_PROFILE=user1**

Windows:

C:\> **set AWS\_PROFILE=user1**

### Set and view configuration settings using commands

<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html#cli-configure-files-where>

There are several ways to view and set your configuration settings using commands.

[**aws configure**](https://docs.aws.amazon.com/cli/latest/reference/configure/index.html)

Run this command to quickly set and view your credentials, Region, and output format. The following example shows sample values.

$ **aws configure**

AWS Access Key ID [None]: ***AKIAIOSFODNN7EXAMPLE***

AWS Secret Access Key [None]: ***wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY***

Default region name [None]: ***us-west-2***

Default output format [None]: ***json***

[**aws configure set**](https://docs.aws.amazon.com/cli/latest/reference/configure/set.html)

You can set any credentials or configuration settings using aws configure set. Specify the profile that you want to view or modify with the --profile setting.

For example, the following command sets the region in the profile named integ.

$ **aws configure set region *us-west-2* --profile *integ***

[**aws configure get**](https://docs.aws.amazon.com/cli/latest/reference/configure/get.html)

You can retrieve any credentials or configuration settings you've set using aws configure get. Specify the profile that you want to view or modify with the --profile setting.

For example, the following command retrieves the region setting in the profile named integ.

$ **aws configure get *region* --profile *integ***

us-west-2

If the output is empty, the setting is not explicitly set and uses the default value.

[**aws configure list**](https://docs.aws.amazon.com/cli/latest/reference/configure/list.html)

To list all configuration data, use the aws configure list command. This command displays the AWS CLI name of all settings you've configured, their values, and where the configuration was retrieved from.

$ **aws configure list**

Name Value Type Location

---- ----- ---- --------

profile <not set> None None

access\_key \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ABCD shared-credentials-file

secret\_key \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ABCD shared-credentials-file

region us-west-2 env AWS\_DEFAULT\_REGION

[**aws configure list-profiles**](https://docs.aws.amazon.com/cli/latest/reference/configure/list-profiles.html)

To list all your profile names, use the aws configure list-profiles command.

$ **aws configure list-profiles**

default

test

### Manually editing the credentials and config files

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html>

When copy and pasting information, we suggest manually editing the config and credentials file. Based on the credential method you prefer, the files are setup in a different way. The following examples show a default profile and a profile named user1 and use sample values. Replace sample values with your own. For more information on the credentials and config files, see [Configuration and credential file settings](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html).

This example is for the short-term credentials from AWS Identity and Access Management. For more information, see [Authenticating using short-term credentials](https://docs.aws.amazon.com/cli/latest/userguide/cli-authentication-short-term.html).

**Credentials file**

[default]

aws\_access\_key\_id=*AKIAIOSFODNN7EXAMPLE*

aws\_secret\_access\_key=*wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*

aws\_session\_token = *IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRINGEXAMPLE*

[user1]

aws\_access\_key\_id=*AKIAI44QH8DHBEXAMPLE*

aws\_secret\_access\_key=*je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY*

aws\_session\_token = *fcZib3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRINGEXAMPLE*

**Config file**

[default]

region=*us-west-2*

output=*json*

[profile user1]

region=*us-east-1*

output=*text*

## Environment Variables

**For CLI**:

* AWS\_ACCESS\_KEY\_ID
* AWS\_SECRET\_ACCESS\_KEY
* AWS\_DEFAULT\_REGION

For NodeJS SDK:

* AWS\_Region

## Create an EC2 Linux Instance

**Syntax**:

aws ec2 run-instances

--image-id ami-xxxxxxxx

--count 1

--instance-type t2.micro

--key-name <key pair name>

--security-group-ids <sg id>

--subnet-id <subnet id>

**Example:**

For Mumbai:

aws ec2 run-instances --image-id ami-06fc49795bc410a0c --count 1 --instance-type t2.micro --key-name MyKeyPairName --security-group-ids sg-0f0ca2490d81252a6 --subnet-id subnet-0246e1175ff243548

For Ohio (*us-east-1*):

aws ec2 run-instances --image-id ami-069aabeee6f53e7bf --count 1 --instance-type t2.micro --key-name MyKeyPairName --security-group-ids sg-0f0ca2490d81252a6 --subnet-id subnet-0246e1175ff243548

For Ohio (*us-east-2*):

aws ec2 run-instances --image-id ami-0b0f111b5dcb2800f --count 1 --instance-type t2.micro --key-name MyKeyPairName --security-group-ids sg-0f0ca2490d81252a6 --subnet-id subnet-0246e1175ff243548

## AWS SDK for JavaScript / NodeJS

<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/getting-started-nodejs.html>

<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sdk-code-samples.html>

### Getting Started in Node.js

#### The Scenario

The example shows how to set up and run a simple Node.js module that creates an Amazon S3 bucket, then adds a text object to it.

Because bucket names in Amazon S3 must be globally unique, this example includes a third-party Node.js module that generates a unique ID value that you can incorporate into the bucket name. This additional module is named uuid.

#### Prerequisite Tasks

To set up and run this example, you must first complete these tasks:

* Create a working directory for developing your Node.js module. Name this directory awsnodesample. Note that the directory must be created in a location that can be updated by applications. For example, in Windows, do not create the directory under "C:\Program Files".
* Install Node.js. For more information, see the [Node.js website](https://nodejs.org/). You can find downloads of the current and LTS versions of Node.js for a variety of operating systems at <https://nodejs.org/en/download/current/>.

#### Step 1: Install the SDK and Dependencies

You install the SDK for JavaScript package using [npm (the Node.js package manager)](https://www.npmjs.com/).

From the awsnodesample directory in the package, type the following at the command line.

npm install aws-sdk

This command installs the SDK for JavaScript in your project, and updates package.json to list the SDK as a project dependency. You can find information about this package by searching for "aws-sdk" on the [npm website](https://www.npmjs.com/).

Next, install the uuid module to the project by typing the following at the command line, which installs the module and updates package.json. For more information about uuid, see the module's page at <https://www.npmjs.com/package/uuid>.

npm install uuid

These packages and their associated code are installed in the node\_modules subdirectory of your project.

#### Step 2: Configure Your Credentials

You need to provide credentials to AWS so that only your account and its resources are accessed by the SDK. For more information about obtaining your account credentials, see [Getting Your Credentials](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/getting-your-credentials.html).

To hold this information, we recommend you create a shared credentials file. To learn how, see [Loading Credentials in Node.js from the Shared Credentials File](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-node-credentials-shared.html). Your credentials file should resemble the following example.

[default]

aws\_access\_key\_id = *YOUR\_ACCESS\_KEY\_ID*

aws\_secret\_access\_key = *YOUR\_SECRET\_ACCESS\_KEY*

You can determine whether you have set your credentials correctly by executing the following code with node:

var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {

if (err) console.log(err.stack);

*// credentials not loaded*

else {

console.log("Access key:", AWS.config.credentials.accessKeyId);

}

});

Similarly, if you have set your region correctly in your config file, you can display that value by setting the AWS\_SDK\_LOAD\_CONFIG environment variable to a truthy value and using the following code:

var AWS = require("aws-sdk");

console.log("Region: ", AWS.config.region);

#### Step 3: Create the Package JSON for the Project

After you create the awsnodesample project directory, you create and add a package.json file for holding the metadata for your Node.js project. For details about using package.json in a Node.js project, see [Creating a package.json file](https://docs.npmjs.com/creating-a-package-json-file).

In the project directory, create a new file named package.json. Then add this JSON to the file.

{

"dependencies": {},

"name": "aws-nodejs-sample",

"description": "A simple Node.js application illustrating usage of the SDK for JavaScript.",

"version": "1.0.1",

"main": "sample.js",

"devDependencies": {},

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "*NAME*",

"license": "*ISC*"

}

Save the file. As you install the modules you need, the dependencies portion of the file will be completed. You can find a JSON file that shows an example of these dependencies [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/blob/main/javascript/example_code/nodegetstarted/example_package.json).

#### Step 4: Write the Node.js Code

Create a new file named sample.js to contain the example code. Begin by adding the require function calls to include the SDK for JavaScript and uuid modules so that they are available for you to use.

Build a unique bucket name that is used to create an Amazon S3 bucket by appending a unique ID value to a recognizable prefix, in this case 'node-sdk-sample-'. You generate the unique ID by calling the uuid module. Then create a name for the Key parameter used to upload an object to the bucket.

Create a promise object to call the createBucket method of the AWS.S3 service object. On a successful response, create the parameters needed to upload text to the newly created bucket. Using another promise, call the putObject method to upload the text object to the bucket.

*// Load the SDK and UUID*

var AWS = require('aws-sdk');

var uuid = require('uuid');

*// Create unique bucket name*

var bucketName = 'node-sdk-sample-' + uuid.v4();

*// Create name for uploaded object key*

var keyName = 'hello\_world.txt';

*// Create a promise on S3 service object*

var bucketPromise = new AWS.S3({apiVersion: '2006-03-01'}).createBucket({Bucket: bucketName}).promise();

*// Handle promise fulfilled/rejected states*

bucketPromise.then(

function(data) {

*// Create params for putObject call*

var objectParams = {Bucket: bucketName, Key: keyName, Body: 'Hello World!'};

*// Create object upload promise*

var uploadPromise = new AWS.S3({apiVersion: '2006-03-01'}).putObject(objectParams).promise();

uploadPromise.then(

function(data) {

console.log("Successfully uploaded data to " + bucketName + "/" + keyName);

});

}).catch(

function(err) {

console.error(err, err.stack);

});

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/blob/main/javascript/example_code/nodegetstarted/sample.js).

#### Step 5: Run the Sample

Type the following command to run the sample.

node sample.js

If the upload is successful, you'll see a confirmation message at the command line. You can also find the bucket and the uploaded text object in the [Amazon S3 console](https://console.aws.amazon.com/s3/).

## Amazon DynamoDB Examples

<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/dynamodb-examples.html>

Amazon DynamoDB is a fully managed NoSQL cloud database that supports both document and key-value store models. You create schemaless tables for data without the need to provision or maintain dedicated database servers.


                Relationship between JavaScript environments, the SDK, and DynamoDB
            

The JavaScript API for DynamoDB is exposed through the AWS.DynamoDB, AWS.DynamoDBStreams, and AWS.DynamoDB.DocumentClient client classes. For more information about using the DynamoDB client classes, see [Class: AWS.DynamoDB](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html), [Class: AWS.DynamoDBStreams](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDBStreams.html), and [Class: AWS.DynamoDB.DocumentClient](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB/DocumentClient.html) in the API reference.

### Creating and Using Tables in DynamoDB

#### The Scenario

Similar to other database systems, DynamoDB stores data in tables. A DynamoDB table is a collection of data that's organized into items that are analogous to rows. To store or access data in DynamoDB, you create and work with tables.

In this example, you use a series of Node.js modules to perform basic operations with a DynamoDB table. The code uses the SDK for JavaScript to create and work with tables by using these methods of the AWS.DynamoDB client class:

* [createTable](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#createTable-property)
* [listTables](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#listTables-property)
* [describeTable](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#describeTable-property)
* [deleteTable](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#deleteTable-property)

#### Prerequisite Tasks

To set up and run this example, first complete these tasks:

* Install Node.js. For more information, see the [Node.js website](https://nodejs.org/).
* Create a shared configurations file with your user credentials. For more information about providing a shared credentials file, see [Loading Credentials in Node.js from the Shared Credentials File](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-node-credentials-shared.html).
* Environment variable for AWS\_REGION or AWS\_DEFAULT\_REGION is existing.

#### Creating a Table

Create a Node.js module with the file name ddb\_createtable.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to create a table, which in this example includes the name and data type for each attribute, the key schema, the name of the table, and the units of throughput to provision. Call the createTable method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

*var AWS = require('aws-sdk');*

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*//AWS.config.update({region: 'REGION'}); // Reads from environment var AWS\_REGION.*

*// Create the DynamoDB service object*

*var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});*

*var params = {*

*AttributeDefinitions: [*

*{*

*AttributeName: 'Artist',*

*AttributeType: 'S'*

*},*

*{*

*AttributeName: 'SongTitle',*

*AttributeType: 'S'*

*}*

*],*

*KeySchema: [*

*{*

*AttributeName: 'Artist',*

*KeyType: 'HASH'*

*},*

*{*

*AttributeName: 'SongTitle',*

*KeyType: 'RANGE'*

*}*

*],*

*ProvisionedThroughput: {*

*ReadCapacityUnits: 1,*

*WriteCapacityUnits: 1*

*},*

*TableName: 'Music',*

*StreamSpecification: {*

*StreamEnabled: false*

*}*

*};*

*// Call DynamoDB to create the table*

*ddb.createTable(params, function(err, data) {*

*if (err) {*

*console.log("Error", err);*

*} else {*

*console.log("Table Created", data);*

*}*

*});*

To run the example, type the following at the command line.

node ddb\_createtable.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_createtable.js).

#### Listing Your Tables

Create a Node.js module with the file name ddb\_listtables.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to list your tables, which in this example limits the number of tables listed to 10. Call the listTables method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

*var AWS = require('aws-sdk');*

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*//AWS.config.update({region: 'REGION'}); // Reads from environment var AWS\_REGION.*

*// Create the DynamoDB service object*

*var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});*

*// Call DynamoDB to retrieve the list of tables*

*ddb.listTables({Limit: 10}, function(err, data) {*

*if (err) {*

*console.log("Error", err.code);*

*} else {*

*console.log("Table names are ", data.TableNames);*

*}*

*});*

To run the example, type the following at the command line.

node ddb\_listtables.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_listtables.js).

#### Describing a Table

Create a Node.js module with the file name ddb\_describetable.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to describe a table, which in this example includes the name of the table provided as a command-line parameter. Call the describeTable method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

var AWS = require('aws-sdk');

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*// AWS.config.update({region: 'REGION'});*

*// Create the DynamoDB service object*

var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {

TableName: process.argv[2]

};

*// Call DynamoDB to retrieve the selected table descriptions*

ddb.describeTable(params, function(err, data) {

if (err) {

console.log("Error", err);

} else {

console.log("Success", data.Table.KeySchema);

}

});

To run the example, type the following at the command line.

node ddb\_describetable.js *CUSTOMER\_LIST*

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_describetable.js).

#### Deleting a Table

Create a Node.js module with the file name ddb\_deletetable.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to delete a table, which in this example includes the name of the table provided as a command-line parameter. Call the deleteTable method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

var AWS = require('aws-sdk');

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*// AWS.config.update({region: 'REGION'});*

*// Create the DynamoDB service object*

var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {

TableName: process.argv[2]

};

*// Call DynamoDB to delete the specified table*

ddb.deleteTable(params, function(err, data) {

if (err && err.code === 'ResourceNotFoundException') {

console.log("Error: Table not found");

} else if (err && err.code === 'ResourceInUseException') {

console.log("Error: Table in use");

} else {

console.log("Success", data);

}

});

To run the example, type the following at the command line.

node ddb\_deletetable.js *CUSTOMER\_LIST*

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_deletetable.js).

### Reading and Writing A Single Item in DynamoDB

<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/dynamodb-example-table-read-write.html>

#### The Scenario

In this example, you use a series of Node.js modules to read and write one item in a DynamoDB table by using these methods of the AWS.DynamoDB client class:

* [putItem](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#putItem-property)
* [getItem](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#getItem-property)
* [deleteItem](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#deleteItem-property)

#### Prerequisite Tasks

To set up and run this example, first complete these tasks:

* Install Node.js. For more information, see the [Node.js website](https://nodejs.org/).
* Create a shared configurations file with your user credentials. For more information about providing a shared credentials file, see [Loading Credentials in Node.js from the Shared Credentials File](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-node-credentials-shared.html).
* Create a DynamoDB table whose items you can access. For more information about creating a DynamoDB table, see [Creating and Using Tables in DynamoDB](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/dynamodb-examples-using-tables.html).

#### Writing an Item

Create a Node.js module with the file name ddb\_putitem.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to add an item, which in this example includes the name of the table and a map that defines the attributes to set and the values for each attribute. Call the putItem method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

*var AWS = require('aws-sdk');*

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*//AWS.config.update({region: 'REGION'}); // Reads from environment var AWS\_REGION.*

*// Create the DynamoDB service object*

*var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});*

*var params = {*

*TableName: 'Music',*

*Item: {*

*'Artist' : {S: 'Led Zeppelin'},*

*'SongTitle' : {S: 'Stairway to Heaven'},*

*'AlbumTitle' : {S: 'Led Zeppelin IV'},*

*'Highlights' : {S: 'Song of the century'}*

*}*

*};*

*// Call DynamoDB to create the item in the table.*

*ddb.putItem(params, function(err, data) {*

*if (err) {*

*console.log("Error: ", err);*

*} else {*

*console.log("Success", data);*

*}*

*});*

To run the example, type the following at the command line.

node ddb\_putitem.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_putitem.js).

#### Getting an Item

Create a Node.js module with the file name ddb\_getitem.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. To identify the item to get, you must provide the value of the primary key for that item in the table. By default, the getItem method returns all the attribute values defined for the item. To get only a subset of all possible attribute values, specify a projection expression.

Create a JSON object containing the parameters needed to get an item, which in this example includes the name of the table, the name and value of the key for the item you're getting, and a projection expression that identifies the item attribute you want to retrieve. Call the getItem method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

*var AWS = require('aws-sdk');*

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*//AWS.config.update({region: 'REGION'}); // Reads from environment var AWS\_REGION.*

*// Create the DynamoDB service object*

*var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});*

*//var ddb = new AWS.DynamoDB.DocumentClient()*

*var params = {*

*TableName: 'Music',*

*Key: {*

*'Artist': {S: 'Post Malone'},*

*'SongTitle' : {S: 'Sunflower'}*

*},*

*ProjectionExpression: 'Artist, SongTitle, AlbumTitle'*

*// ProjectionExpression: 'Artist, SongTitle, AlbumTitle, Highlights'*

*};*

*// Call DynamoDB to create the item in the table.*

*ddb.getItem(params, function(err, data) {*

*if (err) {*

*console.log("Error: ", err);*

*} else {*

*console.log("Success", data);*

*}*

*});*

To run the example, type the following at the command line.

node ddb\_getitem.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_getitem.js).

#### Update an Item

**Note**: Create a new item first.

Create a Node.js module with the file name ddb\_updateitem.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to delete an item, which in this example includes the name of the table and both the key name and value for the item you're deleting. Call the updateItem method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

*var AWS = require('aws-sdk');*

*// Set the region*

*// AWS.config.update({region: 'us-east-2'});*

*//AWS.config.update({region: 'REGION'}); // Reads from environment var AWS\_REGION.*

*// Create the DynamoDB service object*

*var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});*

*var params = {*

*TableName: 'Music',*

*Key: {*

*'Artist': {S: 'Post Malone'},*

*'SongTitle' : {S: 'Sunflower'}*

*},*

*UpdateExpression: "set AlbumTitle = :album",*

*ExpressionAttributeValues : {":album":{"S":"OST - Spiderman: Into the Spider-Verse"}}*

*};*

*// Call DynamoDB to create the item in the table.*

*ddb.updateItem(params, function(err, data) {*

*if (err) {*

*console.log("Error: ", err);*

*} else {*

*console.log("Success", data);*

*}*

*});*

#### Deleting an Item

Create a Node.js module with the file name ddb\_deleteitem.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to delete an item, which in this example includes the name of the table and both the key name and value for the item you're deleting. Call the deleteItem method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

var AWS = require('aws-sdk');

*// Set the region*

AWS.config.update({region: 'REGION'});

*// Create the DynamoDB service object*

var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {

TableName: 'music',

*Key: {*

*'Artist': {S: 'Post Malone'},*

*'SongTitle' : {S: 'Sunflower'}*

*}*

};

*// Call DynamoDB to delete the item from the table*

ddb.deleteItem(params, function(err, data) {

if (err) {

console.log("Error", err);

} else {

console.log("Success", data);

}

});

To run the example, type the following at the command line.

node ddb\_deleteitem.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_deleteitem.js).

### Querying and Scanning a DynamoDB Table

<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/dynamodb-example-query-scan.html>

#### The Scenario

Querying finds items in a table or a secondary index using only primary key attribute values. You must provide a partition key name and a value for which to search. You can also provide a sort key name and value, and use a comparison operator to refine the search results. Scanning finds items by checking every item in the specified table.

In this example, you use a series of Node.js modules to identify one or more items you want to retrieve from a DynamoDB table. The code uses the SDK for JavaScript to query and scan tables using these methods of the DynamoDB client class:

* [query](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#query-property)
* [scan](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#scan-property)

#### Prerequisite Tasks

To set up and run this example, first complete these tasks:

* Install Node.js. For more information, see the [Node.js website](https://nodejs.org/).
* Create a shared configurations file with your user credentials. For more information about providing a shared credentials file, see [Loading Credentials in Node.js from the Shared Credentials File](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-node-credentials-shared.html).
* Create a DynamoDB table whose items you can access. For more information about creating a DynamoDB table, see [Creating and Using Tables in DynamoDB](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/dynamodb-examples-using-tables.html).

#### Querying a Table

This example queries a table that contains episode information about a video series, returning the episode titles and subtitles of second season episodes past episode 9 that contain a specified phrase in their subtitle.

Create a Node.js module with the file name ddb\_query.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to query the table, which in this example includes the table name, the ExpressionAttributeValues needed by the query, a KeyConditionExpression that uses those values to define which items the query returns, and the names of attribute values to return for each item. Call the query method of the DynamoDB service object.

*// Load the AWS SDK for Node.js*

var AWS = require('aws-sdk');

*// Set the region*

AWS.config.update({region: 'REGION'});

*// Create DynamoDB service object*

var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {

ExpressionAttributeValues: {

':s': {N: '2'},

':e' : {N: '09'},

':topic' : {S: 'PHRASE'}

},

KeyConditionExpression: 'Season = :s and Episode > :e',

ProjectionExpression: 'Episode, Title, Subtitle',

FilterExpression: 'contains (Subtitle, :topic)',

TableName: 'EPISODES\_TABLE'

};

ddb.query(params, function(err, data) {

if (err) {

console.log("Error", err);

} else {

*//console.log("Success", data.Items);*

data.Items.forEach(function(element, index, array) {

console.log(element.Title.S + " (" + element.Subtitle.S + ")");

});

}

});

To run the example, type the following at the command line.

node ddb\_query.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_query.js).

#### Scanning a Table

Create a Node.js module with the file name ddb\_scan.js. Be sure to configure the SDK as previously shown. To access DynamoDB, create an AWS.DynamoDB service object. Create a JSON object containing the parameters needed to scan the table for items, which in this example includes the name of the table, the list of attribute values to return for each matching item, and an expression to filter the result set to find items containing a specified phrase. Call the scan method of the DynamoDB service object.

*// Load the AWS SDK for Node.js.*

var AWS = require("aws-sdk");

*// Set the AWS Region.*

AWS.config.update({ region: "REGION" });

*// Create DynamoDB service object.*

var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {

*// Specify which items in the results are returned.*

FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",

*// Define the expression attribute value, which are substitutes for the values you want to compare.*

ExpressionAttributeValues: {

":topic": {S: "SubTitle2"},

":s": {N: 1},

":e": {N: 2},

},

*// Set the projection expression, which are the attributes that you want.*

ProjectionExpression: "Season, Episode, Title, Subtitle",

TableName: "EPISODES\_TABLE",

};

ddb.scan(params, function (err, data) {

if (err) {

console.log("Error", err);

} else {

console.log("Success", data);

data.Items.forEach(function (element, index, array) {

console.log(

"printing",

element.Title.S + " (" + element.Subtitle.S + ")"

);

});

}

});

To run the example, type the following at the command line.

node ddb\_scan.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/dynamodb/ddb_scan.js).

## Amazon EC2 Examples

**This Node.js code example shows:**

* How to create an Amazon EC2 instance from a public Amazon Machine Image (AMI).
* How to create and assign tags to the new Amazon EC2 instance.

### About the Example

In this example, you use a Node.js module to create an Amazon EC2 instance and assign both a key pair and tags to it. The code uses the SDK for JavaScript to create and tag an instance by using these methods of the Amazon EC2 client class:

* [runInstances](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/EC2.html#runInstances-property)
* [createTags](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/EC2.html#createTags-property)

#### Prerequisite Tasks

To set up and run this example, first complete these tasks.

* Install Node.js. For more information, see the [Node.js website](https://nodejs.org/).
* Create a shared configurations file with your user credentials. For more information about providing a shared credentials file, see [Loading Credentials in Node.js from the Shared Credentials File](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-node-credentials-shared.html).
* Create a key pair. For details, see [Working with Amazon EC2 Key Pairs](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/ec2-example-key-pairs.html). You use the name of the key pair in this example.

#### Creating and Tagging an Instance

Create a Node.js module with the file name ec2\_createinstances.js. Be sure to configure the SDK as previously shown.

Create an object to pass the parameters for the runInstances method of the AWS.EC2 client class, including the name of the key pair to assign and the ID of the AMI to run. To call the runInstances method, create a promise for invoking an Amazon EC2 service object, passing the parameters. Then handle the response in the promise callback.

The code next adds a Name tag to a new instance, which the Amazon EC2 console recognizes and displays in the **Name** field of the instance list. You can add up to 50 tags to an instance, all of which can be added in a single call to the createTags method.

*// Load the AWS SDK for Node.js*

var AWS = require('aws-sdk');

*// Load credentials and set region from JSON file*

AWS.config.update({region: 'REGION'});

*// Create EC2 service object*

var ec2 = new AWS.EC2({apiVersion: '2016-11-15'});

*// AMI is amzn-ami-2011.09.1.x86\_64-ebs*

var instanceParams = {

ImageId: 'AMI\_ID',

InstanceType: 't2.micro',

KeyName: 'KEY\_PAIR\_NAME',

MinCount: 1,

MaxCount: 1

};

*// Create a promise on an EC2 service object*

var instancePromise = new AWS.EC2({apiVersion: '2016-11-15'}).runInstances(instanceParams).promise();

*// Handle promise's fulfilled/rejected states*

instancePromise.then(

function(data) {

console.log(data);

var instanceId = data.Instances[0].InstanceId;

console.log("Created instance", instanceId);

*// Add tags to the instance*

tagParams = {Resources: [instanceId], Tags: [

{

Key: 'Name',

Value: 'SDK Sample'

}

]};

*// Create a promise on an EC2 service object*

var tagPromise = new AWS.EC2({apiVersion: '2016-11-15'}).createTags(tagParams).promise();

*// Handle promise's fulfilled/rejected states*

tagPromise.then(

function(data) {

console.log("Instance tagged");

}).catch(

function(err) {

console.error(err, err.stack);

});

}).catch(

function(err) {

console.error(err, err.stack);

});

To run the example, type the following at the command line.

node ec2\_createinstances.js

This sample code can be found [here on GitHub](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example_code/ec2/ec2_createinstances.js).

## AWS SDK for Python

**EC2:**

<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html>

<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html>

<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/ec2-example-managing-instances.html>

**S3:**

<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-example-creating-buckets.html>

<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-uploading-files.html>

<https://towardsdatascience.com/introduction-to-pythons-boto3-c5ac2a86bb63>

**DynamoDB:**

<https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html>

# AWS DevOps – CodeCommit, CodeBuild, CodeDeploy, CodePipeline, CodeStar

<https://medium.com/beeranddiapers/how-to-create-a-aws-code-pipeline-using-aws-code-commit-code-build-and-code-deploy-8c555a6401fc>

<https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-simple-s3.html#S3-create-deployment>

<https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/>

**AWS CodeDeploy vs AWS CodePipeline: What are the differences?**

**AWS CodeDeploy:** Coordinate application deployments to Amazon EC2 instances. AWS CodeDeploy is a service that automates code deployments to Amazon EC2 instances. AWS CodeDeploy makes it easier for you to rapidly release new features, helps you avoid downtime during deployment, and handles the complexity of updating your applications; **AWS CodePipeline:** Continuous delivery service for fast and reliable application updates. CodePipeline builds, tests, and deploys your code every time there is a code change, based on the release process models you define.

AWS CodeDeploy belongs to **"Deployment as a Service"** category of the tech stack, while AWS CodePipeline can be primarily classified under **"Continuous Deployment"**.

**CodeStar:**

<https://docs.aws.amazon.com/codestar/latest/userguide/welcome.html>

<https://docs.aws.amazon.com/codestar/latest/userguide/setting-up.html>

<https://docs.aws.amazon.com/codestar/latest/userguide/getting-started.html>

<https://github.com/Lucasbatistasilveira/petpipeline>

# Deployment and Provisioning

## Amazon Elastic Beanstalk

### What is AWS Elastic Beanstalk?

Amazon Web Services (AWS) comprises over one hundred services, each of which exposes an area of functionality. While the variety of services offers flexibility for how you want to manage your AWS infrastructure, it can be challenging to figure out which services to use and how to provision them.

With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without having to learn about the infrastructure that runs those applications. Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

Elastic Beanstalk supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby. When you deploy your application, Elastic Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, to run your application.

You can interact with Elastic Beanstalk by using the Elastic Beanstalk console, the AWS Command Line Interface (AWS CLI), or **eb**, a high-level CLI designed specifically for Elastic Beanstalk.

To learn more about how to deploy a sample web application using Elastic Beanstalk, see [Getting Started with AWS: Deploying a Web App](https://docs.aws.amazon.com/gettingstarted/latest/deploy/).

You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the Elastic Beanstalk web interface (console).

To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions. The following diagram illustrates the workflow of Elastic Beanstalk.


        Elastic Beanstalk flow
      

After you create and deploy your application, information about the application—including metrics, events, and environment status—is available through the Elastic Beanstalk console, APIs, or Command Line Interfaces, including the unified AWS CLI.

### Elastic Beanstalk concepts

AWS Elastic Beanstalk enables you to manage all of the resources that run your application as environments. Here are some key Elastic Beanstalk concepts.

#### Application

An Elastic Beanstalk application is a logical collection of Elastic Beanstalk components, including environments, versions, and environment configurations. In Elastic Beanstalk an application is conceptually similar to a folder.

#### Application version

In Elastic Beanstalk, an application version refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon Simple Storage Service (Amazon S3) object that contains the deployable code, such as a Java WAR file. An application version is part of an application. Applications can have many versions and each application version is unique. In a running environment, you can deploy any application version you already uploaded to the application, or you can upload and immediately deploy a new application version. You might upload multiple application versions to test differences between one version of your web application and another.

#### Environment

An environment is a collection of AWS resources running an application version. Each environment runs only one application version at a time, however, you can run the same application version or different application versions in many environments simultaneously. When you create an environment, Elastic Beanstalk provisions the resources needed to run the application version you specified.

#### Environment tier

When you launch an Elastic Beanstalk environment, you first choose an environment tier. The environment tier designates the type of application that the environment runs, and determines what resources Elastic Beanstalk provisions to support it. An application that serves HTTP requests runs in a [web server environment tier](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-webserver.html). A backend environment that pulls tasks from an Amazon Simple Queue Service (Amazon SQS) queue runs in a [worker environment tier](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-worker.html).

#### Environment configuration

An environment configuration identifies a collection of parameters and settings that define how an environment and its associated resources behave. When you update an environment’s configuration settings, Elastic Beanstalk automatically applies the changes to existing resources or deletes and deploys new resources (depending on the type of change).

#### Saved configuration

A saved configuration is a template that you can use as a starting point for creating unique environment configurations. You can create and modify saved configurations, and apply them to environments, using the Elastic Beanstalk console, EB CLI, AWS CLI, or API. The API and the AWS CLI refer to saved configurations as configuration templates.

#### Platform

A platform is a combination of an operating system, programming language runtime, web server, application server, and Elastic Beanstalk components. You design and target your web application to a platform. Elastic Beanstalk provides a variety of platforms on which you can build your applications.

### Web server environments

The following diagram shows an example Elastic Beanstalk architecture for a web server environment tier, and shows how the components in that type of environment tier work together.


      AWS Elastic Beanstalk architecture diagram
    

The environment is the heart of the application. In the diagram, the environment is shown within the top-level solid line. When you create an environment, Elastic Beanstalk provisions the resources required to run your application. AWS resources created for an environment include one elastic load balancer (ELB in the diagram), an Auto Scaling group, and one or more Amazon Elastic Compute Cloud (Amazon EC2) instances.

Every environment has a CNAME (URL) that points to a load balancer. The environment has a URL, such as myapp.us-west-2.elasticbeanstalk.com. This URL is aliased in [Amazon Route 53](https://aws.amazon.com/route53/) to an Elastic Load Balancing URL—something like abcdef-123456.us-west-2.elb.amazonaws.com—by using a CNAME record. [Amazon Route 53](https://aws.amazon.com/route53/) is a highly available and scalable Domain Name System (DNS) web service. It provides secure and reliable routing to your infrastructure. Your domain name that you registered with your DNS provider will forward requests to the CNAME.

The load balancer sits in front of the Amazon EC2 instances, which are part of an Auto Scaling group. Amazon EC2 Auto Scaling automatically starts additional Amazon EC2 instances to accommodate increasing load on your application. If the load on your application decreases, Amazon EC2 Auto Scaling stops instances, but always leaves at least one instance running.

The software stack running on the Amazon EC2 instances is dependent on the container type. A container type defines the infrastructure topology and software stack to be used for that environment. For example, an Elastic Beanstalk environment with an Apache Tomcat container uses the Amazon Linux operating system, Apache web server, and Apache Tomcat software. For a list of supported container types, see [Elastic Beanstalk supported platforms](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html). Each Amazon EC2 instance that runs your application uses one of these container types. In addition, a software component called the host manager (HM) runs on each Amazon EC2 instance. The host manager is responsible for the following:

* Deploying the application
* Aggregating events and metrics for retrieval via the console, the API, or the command line
* Generating instance-level events
* Monitoring the application log files for critical errors
* Monitoring the application server
* Patching instance components
* Rotating your application's log files and publishing them to Amazon S3

The host manager reports metrics, errors and events, and server instance status, which are available via the Elastic Beanstalk console, APIs, and CLIs.

The Amazon EC2 instances shown in the diagram are part of one security group. A security group defines the firewall rules for your instances. By default, Elastic Beanstalk defines a security group, which allows everyone to connect using port 80 (HTTP). You can define more than one security group. For example, you can define a security group for your database server.

### Worker environments

AWS resources created for a worker environment tier include an Auto Scaling group, one or more Amazon EC2 instances, and an IAM role. For the worker environment tier, Elastic Beanstalk also creates and provisions an Amazon SQS queue if you don’t already have one. When you launch a worker environment, Elastic Beanstalk installs the necessary support files for your programming language of choice and a daemon on each EC2 instance in the Auto Scaling group. The daemon reads messages from an Amazon SQS queue. The daemon sends data from each message that it reads to the web application running in the worker environment for processing. If you have multiple instances in your worker environment, each instance has its own daemon, but they all read from the same Amazon SQS queue.

The following diagram shows the different components and their interactions across environments and AWS services.


      AWS Elastic Beanstalk worker tier architecture diagram
    

Amazon CloudWatch is used for alarms and health monitoring. For more information, go to [Basic health reporting](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.healthstatus.html).

For details about how the worker environment tier works, see [Elastic Beanstalk worker environments](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html).

**Demo**:

* Navigate to Elastic Beanstalk.
* Create Application.
* App name.
* Platform -> .NET Core on Linux.
* Sample Application.
* Create Application.
* Wait for some time.
* Select Elastic Beanstalk -> Environments.
* Shows a new env created for the app.
* Open url for the env in a new tab -> Shows sample app.
* Select Elastic Beanstalk -> Applications.
* Shows app created.
* Click on the app name -> env name.
* Shows details.
* Navigate options from left panel.
* Navigate to EC2 -> Instances.
* Applications -> select app -> Actions -> Delete application.
* Environments -> select env -> Actions -> Terminate environment.

## AWS CloudFormation

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>

AWS CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and CloudFormation takes care of provisioning and configuring those resources for you. You don't need to individually create and configure AWS resources and figure out what's dependent on what; CloudFormation handles that. The following scenarios demonstrate how CloudFormation can help.

## Simplify infrastructure management

For a scalable web application that also includes a backend database, you might use an Auto Scaling group, an Elastic Load Balancing load balancer, and an Amazon Relational Database Service database instance. You might use each individual service to provision these resources and after you create the resources, you would have to configure them to work together. All these tasks can add complexity and time before you even get your application up and running.

Instead, you can create a CloudFormation template or modify an existing one. A template describes all your resources and their properties. When you use that template to create a CloudFormation stack, CloudFormation provisions the Auto Scaling group, load balancer, and database for you. After the stack has been successfully created, your AWS resources are up and running. You can delete the stack just as easily, which deletes all the resources in the stack. By using CloudFormation, you easily manage a collection of resources as a single unit.

## Quickly replicate your infrastructure

If your application requires additional availability, you might replicate it in multiple regions so that if one region becomes unavailable, your users can still use your application in other regions. The challenge in replicating your application is that it also requires you to replicate your resources. Not only do you need to record all the resources that your application requires, but you must also provision and configure those resources in each region.

Reuse your CloudFormation template to create your resources in a consistent and repeatable manner. To reuse your template, describe your resources once and then provision the same resources over and over in multiple regions.

## Easily control and track changes to your infrastructure

In some cases, you might have underlying resources that you want to upgrade incrementally. For example, you might change to a higher performing instance type in your Auto Scaling launch configuration so that you can reduce the maximum number of instances in your Auto Scaling group. If problems occur after you complete the update, you might need to roll back your infrastructure to the original settings. To do this manually, you not only have to remember which resources were changed, you also have to know what the original settings were.

When you provision your infrastructure with CloudFormation, the CloudFormation template describes exactly what resources are provisioned and their settings. Because these templates are text files, you simply track differences in your templates to track changes to your infrastructure, similar to the way developers control revisions to source code. For example, you can use a version control system with your templates so that you know exactly what changes were made, who made them, and when. If at any point you need to reverse changes to your infrastructure, you can use a previous version of your template.

## AWS CloudFormation concepts

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-concepts.html>

When you use AWS CloudFormation, you work with templates and stacks. You create templates to describe your AWS resources and their properties. Whenever you create a stack, CloudFormation provisions the resources that are described in your template.

**Topics**

* [Templates](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-concepts.html#cfn-concepts-templates)
* [Stacks](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-concepts.html#w2ab1b5c15b9)
* [Change sets](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-concepts.html#cfn-concepts-change-sets)

### Templates

A CloudFormation template is a JSON or YAML formatted text file. You can save these files with any extension, such as .json, .yaml, .template, or .txt. CloudFormation uses these templates as blueprints for building your AWS resources. For example, in a template, you can describe an Amazon EC2 instance, such as the instance type, the AMI ID, block device mappings, and its Amazon EC2 key pair name. Whenever you create a stack, you also specify a template that CloudFormation uses to create whatever you described in the template.

For example, if you created a stack with the following template, CloudFormation provisions an instance with an ami-0ff8a91507f77f867 AMI ID, t2.micro instance type, testkey key pair name, and an Amazon EBS volume.

**JSON (CloudFormation - Sample JSON 01.json)**

{

"AWSTemplateFormatVersion": "2010-09-09",

"Description": "A sample template",

"Resources": {

"MyEC2Instance": {

"Type": "AWS::EC2::Instance",

"Properties": {

"ImageId": "ami-0ff8a91507f77f867",

"InstanceType": "t2.micro",

"KeyName": "testkey",

"BlockDeviceMappings": [

{

"DeviceName": "/dev/sdm",

"Ebs": {

"VolumeType": "io1",

"Iops": 200,

"DeleteOnTermination": false,

"VolumeSize": 20

}

}

]

}

}

}

}

**YAML (CloudFormation - Sample YAML 01.yml)**

AWSTemplateFormatVersion: 2010-09-09

Description: A sample template

Resources:

MyEC2Instance:

Type: 'AWS::EC2::Instance'

Properties:

ImageId: ami-0ff8a91507f77f867

InstanceType: t2.micro

KeyName: testkey

BlockDeviceMappings:

- DeviceName: /dev/sdm

Ebs:

VolumeType: io1

Iops: 200

DeleteOnTermination: false

VolumeSize: 20

You can also specify multiple resources in a single template and configure these resources to work together. For example, you can modify the previous template to include an Elastic IP address (EIP) and associate it with the Amazon EC2 instance, as shown in the following example:

**JSON**

{

"AWSTemplateFormatVersion": "2010-09-09",

"Description": "A sample template",

"Resources": {

"MyEC2Instance": {

"Type": "AWS::EC2::Instance",

"Properties": {

"ImageId": "ami-0ff8a91507f77f867",

"InstanceType": "t2.micro",

"KeyName": "testkey",

"BlockDeviceMappings": [

{

"DeviceName": "/dev/sdm",

"Ebs": {

"VolumeType": "io1",

"Iops": 200,

"DeleteOnTermination": false,

"VolumeSize": 20

}

}

]

}

},

"MyEIP": {

"Type": "AWS::EC2::EIP",

"Properties": {

"InstanceId": {

"Ref": "MyEC2Instance"

}

}

}

}

}

**YAML**

AWSTemplateFormatVersion: 2010-09-09

Description: A sample template

Resources:

MyEC2Instance:

Type: 'AWS::EC2::Instance'

Properties:

ImageId: ami-0ff8a91507f77f867

InstanceType: t2.micro

KeyName: testkey

BlockDeviceMappings:

- DeviceName: /dev/sdm

Ebs:

VolumeType: io1

Iops: 200

DeleteOnTermination: false

VolumeSize: 20

MyEIP:

Type: 'AWS::EC2::EIP'

Properties:

InstanceId: !Ref MyEC2Instance

The previous templates are centered around a single Amazon EC2 instance; however, CloudFormation templates have additional capabilities that you can use to build complex sets of resources and reuse those templates in multiple contexts. For example, you can add input parameters whose values are specified when you create a CloudFormation stack. In other words, you can specify a value like the instance type when you create a stack instead of when you create the template, making the template easier to reuse in different situations.

### Stacks

When you use CloudFormation, you manage related resources as a single unit called a stack. You create, update, and delete a collection of resources by creating, updating, and deleting stacks. All the resources in a stack are defined by the stack's CloudFormation template. Suppose you created a template that includes an Auto Scaling group, Elastic Load Balancing load balancer, and an Amazon Relational Database Service (Amazon RDS) database instance. To create those resources, you create a stack by submitting the template that you created, and CloudFormation provisions all those resources for you. You can work with stacks by using the CloudFormation [console](https://console.aws.amazon.com/cloudformation/), [API](https://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/), or [AWS CLI](https://docs.aws.amazon.com/cli/latest/reference/cloudformation).

For more information about creating, updating, or deleting stacks, see [Working with stacks](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html).

### Change sets

If you need to make changes to the running resources in a stack, you update the stack. Before making changes to your resources, you can generate a change set, which is a summary of your proposed changes. Change sets allow you to see how your changes might impact your running resources, especially for critical resources, before implementing them.

For example, if you change the name of an Amazon RDS database instance, CloudFormation will create a new database and delete the old one. You will lose the data in the old database unless you've already backed it up. If you generate a change set, you will see that your change will cause your database to be replaced, and you will be able to plan accordingly before you update your stack. For more information, see [Updating stacks using change sets](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html).

## Installation of LAMP Server in EC2 through CloudFormation

1. Navigate to CloudFormation -> Create Stack
2. Select Use a sample template
3. Select LAMP
4. View in Designer or copy S3 url and open in new browser
5. Next
6. Stack name
7. DB name
8. DB Pasword (alphanumeric)
9. DB Root Password (alphanumeric)
10. Key pair
11. Next, Next, Next, Create
12. Wait for a while to create the stack
13. On the stack page, select Outputs tab
14. Open the WebSiteURL in a new window (shows PHP is working).
15. Delete stack.

## Create a VPC with Public and Private Subnets

**AWS CloudFormation VPC Template.yml**

AWSTemplateFormatVersion: 2010-09-09

Description: Deploy a VPC with public / private subnets.

Resources:

VPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: 10.0.0.0/16

EnableDnsHostnames: true

Tags:

- Key: Name

Value: VPC-ajs-4apr

InternetGateway:

Type: AWS::EC2::InternetGateway

Properties:

Tags:

- Key: Name

Value: igw-ajs-4apr

AttachGateway:

Type: AWS::EC2::VPCGatewayAttachment

Properties:

VpcId: !Ref VPC

InternetGatewayId: !Ref InternetGateway

PublicSubnet1:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref VPC

CidrBlock: 10.0.1.0/24

AvailabilityZone: !Select [ 0, !GetAZs '' ]

Tags:

- Key: Name

Value: Public-Subnet-4apr

PrivateSubnet1:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref VPC

CidrBlock: 10.0.2.0/24

AvailabilityZone: !Select [ 0, !GetAZs '' ]

Tags:

- Key: Name

Value: Private-Subnet-4apr

PublicRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

- Key: Name

Value: Public-RT

PublicRoute:

Type: AWS::EC2::Route

DependsOn: AttachGateway

Properties:

RouteTableId: !Ref PublicRouteTable

DestinationCidrBlock: 0.0.0.0/0

GatewayId: !Ref InternetGateway

PublicSubnetRouteTableAssociation1:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

SubnetId: !Ref PublicSubnet1

RouteTableId: !Ref PublicRouteTable

PrivateRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags:

- Key: Name

Value: Private-RT

PrivateSubnetRouteTableAssociation1:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

SubnetId: !Ref PrivateSubnet1

RouteTableId: !Ref PrivateRouteTable

Outputs:

VPC:

Description: VPC

Value: !Ref VPC

AZ1:

Description: Availability Zone 1

Value: !GetAtt

- PublicSubnet1

- AvailabilityZone

1. Navigate to CloudFormation -> Create Stack
2. Select Use an existing template
3. Select “AWS CloudFormation VPC Template.yml”
4. Next -> Next -> Create
5. Do not delete stack as its resources are required for creating an EC2 instance.

## Launch an EC2 Instance in a VPC

<https://cloudkatha.com/how-to-launch-an-ec2-instance-in-an-existing-vpc-using-cloudformation/>

### Querying for the latest AMI using public parameters

**Bash:**

aws ssm get-parameters --names /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2 --region us-east-1

**Powershell:**

Get-SSMParameter -Name /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2 -region us-east-1

**AWS CloudFormation EC2 Instance in an Existing VPC.yml**

AWSTemplateFormatVersion: '2010-09-09'

Description: Template to Create an EC2 instance in a VPC

Parameters:

ImageId:

Type: String

Description: 'Linux 2 AMI for us-east-1 Region'

Default: 'ami-06d3b5e1ed9e1d982'

VpcId:

Type: String

Description: VPC id

Default: <vpc-id>

SubnetId:

Type: String

Description: Subnet in which to launch an EC2

Default: <subnet-id>

AvailabilityZone:

Type: String

Description: Availability Zone into which instance will launch

Default: us-east-1a

InstanceType:

Type: String

Description: Choosing t2 micro because it is free

Default: t2.micro

KeyName:

Description: SSH Keypair to login to the instance

Type: AWS::EC2::KeyPair::KeyName

Default: ajs-us-east-1-key-pair

Resources:

DemoInstance:

Type: 'AWS::EC2::Instance'

Properties:

ImageId: !Ref ImageId

InstanceType: !Ref InstanceType

AvailabilityZone: !Ref AvailabilityZone

KeyName: !Ref KeyName

NetworkInterfaces:

- DeviceIndex: 0

AssociatePublicIpAddress: true

DeleteOnTermination: true

SubnetId: !Ref SubnetId

GroupSet:

- !Ref DemoSecurityGroup

DemoSecurityGroup:

Type: 'AWS::EC2::SecurityGroup'

Properties:

VpcId: !Ref VpcId

GroupDescription: SG to allow SSH access via port 22

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: '22'

ToPort: '22'

CidrIp: '0.0.0.0/0'

Tags:

- Key: Name

Value: sg-tcp-04apr

Outputs:

DemoInstanceId:

Description: Instance Id

Value: !Ref DemoInstance

1. Ensure VPC is created (*refer to section* [*Create a VPC with Public and Private Subnets*](#_Create_a_VPC_1)).
2. Navigate to CloudFormation -> Create Stack
3. Select Use an existing template
4. Select “AWS CloudFormation EC2 Instance in an Existing VPC.yml”
5. Next -> Next -> Create
6. Navigate to EC2 -> Instances and check if instance created.
7. Connect to the instance once ready.
8. Delete stack (*also delete VPC stack created earlier*).

## Walkthrough: Create a scaled and load-balanced application

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/example-templates-autoscaling.html>

For this walkthrough, you create a stack that helps you set up a scaled and load-balanced application. The walkthrough provides a sample template that you use to create the stack. The example template provisions an Auto Scaling group, an Application Load Balancer, security groups that control traffic to the load balancer and to the Auto Scaling group, and an Amazon SNS notification configuration to publish notifications about scaling activities.

This template creates one or more Amazon EC2 instances and an Application Load Balancer. You will be billed for the AWS resources used if you create a stack from this template.

### Full stack template

Let's start with the template (**sampleloadbalancedappstack.yml**).

**YAML**

AWSTemplateFormatVersion: 2010-09-09

Parameters:

InstanceType:

Description: The EC2 instance type

Type: String

Default: t3.micro

AllowedValues:

- t3.micro

- t3.small

- t3.medium

KeyName:

Description: Name of an existing EC2 key pair to allow SSH access to the instances

Type: 'AWS::EC2::KeyPair::KeyName'

LatestAmiId:

Description: The latest Amazon Linux 2 AMI from the Parameter Store

Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'

Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2'

OperatorEmail:

Description: The email address to notify when there are any scaling activities

Type: String

SSHLocation:

Description: The IP address range that can be used to SSH to the EC2 instances

Type: String

MinLength: 9

MaxLength: 18

Default: 0.0.0.0/0

ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

Subnets:

Type: 'List<AWS::EC2::Subnet::Id>'

Description: At least two public subnets in different Availability Zones in the selected VPC

VPC:

Type: 'AWS::EC2::VPC::Id'

Description: A virtual private cloud (VPC) that enables resources in public subnets to connect to the internet

Resources:

ELBSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: ELB Security Group

VpcId: !Ref VPC

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 80

ToPort: 80

CidrIp: 0.0.0.0/0

EC2SecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: EC2 Security Group

VpcId: !Ref VPC

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 80

ToPort: 80

SourceSecurityGroupId:

Fn::GetAtt:

- ELBSecurityGroup

- GroupId

- IpProtocol: tcp

FromPort: 22

ToPort: 22

CidrIp: !Ref SSHLocation

EC2TargetGroup:

Type: AWS::ElasticLoadBalancingV2::TargetGroup

Properties:

HealthCheckIntervalSeconds: 30

HealthCheckProtocol: HTTP

HealthCheckTimeoutSeconds: 15

HealthyThresholdCount: 5

Matcher:

HttpCode: '200'

Name: EC2TargetGroup

Port: 80

Protocol: HTTP

TargetGroupAttributes:

- Key: deregistration\_delay.timeout\_seconds

Value: '20'

UnhealthyThresholdCount: 3

VpcId: !Ref VPC

ALBListener:

Type: AWS::ElasticLoadBalancingV2::Listener

Properties:

DefaultActions:

- Type: forward

TargetGroupArn: !Ref EC2TargetGroup

LoadBalancerArn: !Ref ApplicationLoadBalancer

Port: 80

Protocol: HTTP

ApplicationLoadBalancer:

Type: AWS::ElasticLoadBalancingV2::LoadBalancer

Properties:

Scheme: internet-facing

Subnets: !Ref Subnets

SecurityGroups:

- !GetAtt ELBSecurityGroup.GroupId

LaunchTemplate:

Type: AWS::EC2::LaunchTemplate

Properties:

LaunchTemplateName: !Sub ${AWS::StackName}-launch-template

LaunchTemplateData:

ImageId: !Ref LatestAmiId

InstanceType: !Ref InstanceType

KeyName: !Ref KeyName

SecurityGroupIds:

- !Ref EC2SecurityGroup

UserData:

Fn::Base64: !Sub |

#!/bin/bash

yum update -y

yum install -y httpd

systemctl start httpd

systemctl enable httpd

echo "<h1>Hello World!</h1>" > /var/www/html/index.html

NotificationTopic:

Type: AWS::SNS::Topic

Properties:

Subscription:

- Endpoint: !Ref OperatorEmail

Protocol: email

WebServerGroup:

Type: AWS::AutoScaling::AutoScalingGroup

Properties:

LaunchTemplate:

LaunchTemplateId: !Ref LaunchTemplate

Version: !GetAtt LaunchTemplate.LatestVersionNumber

MaxSize: '3'

MinSize: '1'

NotificationConfigurations:

- TopicARN: !Ref NotificationTopic

NotificationTypes: ['autoscaling:EC2\_INSTANCE\_LAUNCH', 'autoscaling:EC2\_INSTANCE\_LAUNCH\_ERROR', 'autoscaling:EC2\_INSTANCE\_TERMINATE', 'autoscaling:EC2\_INSTANCE\_TERMINATE\_ERROR']

TargetGroupARNs:

- !Ref EC2TargetGroup

VPCZoneIdentifier: !Ref Subnets

### Template walkthrough

The first part of this template specifies the Parameters. Each parameter must be assigned a value at runtime for AWS CloudFormation to successfully provision the stack. Resources specified later in the template reference these values and use the data.

* InstanceType: The type of EC2 instance that Amazon EC2 Auto Scaling provisions. If not specified, a default of t3.micro is used.
* KeyName: An existing EC2 key pair to allow SSH access to the instances.
* LatestAmiId: The Amazon Machine Image (AMI) for the instances. If not specified, your instances are launched with an Amazon Linux 2 AMI, using an AWS Systems Manager public parameter maintained by AWS. For more information, see [Finding public parameters](https://docs.aws.amazon.com/systems-manager/latest/userguide/parameter-store-finding-public-parameters.html) in the AWS Systems Manager User Guide.
* OperatorEmail: The email address where you want to send scaling activity notifications.
* SSHLocation: The IP address range that can be used to SSH to the instances.
* Subnets: At least two public subnets in different Availability Zones.
* VPC: A virtual private cloud (VPC) in your account that enables resources in public subnets to connect to the internet.

**Note**: You can use the default VPC and default subnets to allow instances to access the internet. 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.

The next part of this template specifies the Resources. This section specifies the stack resources and their properties.

[AWS::EC2::SecurityGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group.html) resource ELBSecurityGroup

* SecurityGroupIngress contains a TCP ingress rule that allows access from all IP addresses ("CidrIp" : "0.0.0.0/0") on port 80.

[AWS::EC2::SecurityGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group.html) resource EC2SecurityGroup

* SecurityGroupIngress contains two ingress rules: 1) a TCP ingress rule that allows SSH access (port 22) from the IP address range that you provide for the SSHLocation input parameter and 2) a TCP ingress rule that allows access from the load balancer by specifying the load balancer's security group. The [GetAtt](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference-getatt.html) function is used to get the ID of the security group with the logical name ELBSecurityGroup.

[AWS::ElasticLoadBalancingV2::TargetGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-targetgroup.html) resource EC2TargetGroup

* Port, Protocol, and HealthCheckProtocol specify the EC2 instance port (80) and protocol (HTTP) that the ApplicationLoadBalancer routes traffic to and that Elastic Load Balancing uses to check the health of the EC2 instances.
* HealthCheckIntervalSeconds specifies that the EC2 instances have an interval of 30 seconds between health checks. The HealthCheckTimeoutSeconds is defined as the length of time Elastic Load Balancing waits for a response from the health check target (15 seconds in this example). After the timeout period lapses, Elastic Load Balancing marks that EC2 instance's health check as unhealthy. When an EC2 instance fails three consecutive health checks (UnhealthyThresholdCount), Elastic Load Balancing stops routing traffic to that EC2 instance until that instance has five consecutive healthy health checks (HealthyThresholdCount). At that point, Elastic Load Balancing considers the instance healthy and begins routing traffic to the instance again.
* TargetGroupAttributes updates the deregistration delay value of the target group to 20 seconds. By default, Elastic Load Balancing waits 300 seconds before completing the deregistration process.

[AWS::ElasticLoadBalancingV2::Listener](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-listener.html) resource ALBListener

* DefaultActions specifies the port that the load balancer listens to, the target group where the load balancer forwards requests, and the protocol used to route requests.

[AWS::ElasticLoadBalancingV2::LoadBalancer](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-loadbalancer.html) resource ApplicationLoadBalancer

* Subnets takes the value of the Subnets input parameter as the list of public subnets where the load balancer nodes will be created.
* SecurityGroup gets the ID of the security group that acts as a virtual firewall for your load balancer nodes to control incoming traffic. The [GetAtt](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference-getatt.html) function is used to get the ID of the security group with the logical name ELBSecurityGroup.

[AWS::EC2::LaunchTemplate](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-launchtemplate.html) resource LaunchTemplate

* ImageId takes the value of the LatestAmiId input parameter as the AMI to use.
* KeyName takes the value of the KeyName input parameter as the EC2 key pair to use.
* SecurityGroupIds gets the ID of the security group with the logical name EC2SecurityGroup that acts as a virtual firewall for your EC2 instances to control incoming traffic.
* UserData is a configuration script that runs after the instance is up and running. In this example, the script installs Apache and creates an index.html file.

[AWS::SNS::Topic](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-sns-topic.html) resource NotificationTopic

* Subscription takes the value of the OperatorEmail input parameter as the email address for the recipient of the notifications when there are any scaling activities.

[AWS::AutoScaling::AutoScalingGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-group.html) resource WebServerGroup

* MinSize and MaxSize set the minimum and maximum number of EC2 instances in the Auto Scaling group.
* TargetGroupARNs takes the ARN of the target group with the logical name EC2TargetGroup. As this Auto Scaling group scales, it automatically registers and deregisters instances with this target group.
* VPCZoneIdentifier takes the value of the Subnets input parameter as the list of public subnets where the EC2 instances can be created.

### Step 1: Launch the stack

Before you launch the stack, check that you have AWS Identity and Access Management (IAM) permissions to use all of the following services: Amazon EC2, Amazon EC2 Auto Scaling, AWS Systems Manager, Elastic Load Balancing, Amazon SNS, and AWS CloudFormation.

The following procedure involves uploading the sample stack template from a file. Open a text editor on your local machine and add one of the templates. Save the file with the name sampleloadbalancedappstack.yml.

**To launch the stack template**

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at [https://console.aws.amazon.com/cloudformation](https://console.aws.amazon.com/cloudformation/).
2. Choose **Create stack**, **With new resources (standard)**.
3. Under **Specify template**, choose **Upload a template file**, **Choose file** to upload the sampleloadbalancedappstack.yml file.
4. Choose **Next**.
5. On the **Specify stack details** page, type the stack name (for example, **SampleLoadBalancedAppStack**).
6. Under **Parameters**, review the parameters for the stack and provide values for all parameters that don't have default values, including **OperatorEmail**, **SSHLocation**, **KeyName**, **VPC**, and **Subnets**.
7. Choose **Next** twice.
8. On the **Review** page, review and confirm the settings.
9. Choose **Submit**.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. When AWS CloudFormation has successfully created the stack, you receive a status of **CREATE\_COMPLETE**.

**Note**: After you create the stack, you must confirm the subscription before the email address can start to receive notifications. For more information, see [Get Amazon SNS notifications when your Auto Scaling group scales](https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-sns-notifications.html) in the Amazon EC2 Auto Scaling User Guide.

### Step 2: Clean up your sample resources

To make sure that you aren't charged for unused sample resources, delete the stack.

To delete the stack

1. In the AWS CloudFormation console, select the **SampleLoadBalancedAppStack** stack.
2. Choose **Delete**.
3. In the confirmation message, choose **Delete stack**.
4. The status for **SampleLoadBalancedAppStack** changes to **DELETE\_IN\_PROGRESS**. When AWS CloudFormation completes the deletion of the stack, it removes the stack from the list.

# AWS Cloud Development Kit (CDK)

<https://docs.aws.amazon.com/cdk/v2/guide/home.html>

Welcome to the AWS Cloud Development Kit (AWS CDK) Developer Guide. This document provides information about the AWS CDK, a framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation.

**Note**: The CDK has been released in two major versions, v1 and v2. This is the Developer Guide for AWS CDK v2. The earlier CDK v1 entered maintenance on June 1, 2022. Support for CDK v1 will end on June 1, 2023.

The AWS CDK lets you build reliable, scalable, cost-effective applications in the cloud with the considerable expressive power of a programming language. This approach yields many benefits, including:

Build with high-level constructs that automatically provide sensible, secure defaults for your AWS resources, defining more infrastructure with less code.

Use programming idioms like parameters, conditionals, loops, composition, and inheritance to model your system design from building blocks provided by AWS and others.

Put your infrastructure, application code, and configuration all in one place, ensuring that at every milestone you have a complete, cloud-deployable system.

Employ software engineering practices such as code reviews, unit tests, and source control to make your infrastructure more robust.

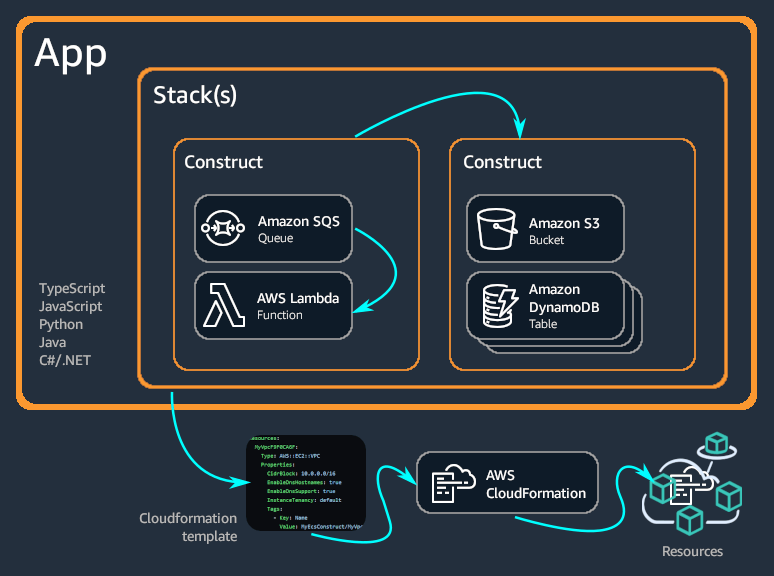
Connect your AWS resources together (even across stacks) and grant permissions using simple, intent-oriented APIs.

Import existing AWS CloudFormation templates to give your resources a CDK API.

Use the power of AWS CloudFormation to perform infrastructure deployments predictably and repeatedly, with rollback on error.

Easily share infrastructure design patterns among teams within your organization or even with the public.

The AWS CDK supports TypeScript, JavaScript, Python, Java, C#/.Net, and Go. Developers can use one of these supported programming languages to define reusable cloud components known as [Constructs](https://docs.aws.amazon.com/cdk/v2/guide/constructs.html). You compose these together into [Stacks](https://docs.aws.amazon.com/cdk/v2/guide/stacks.html) and [Apps](https://docs.aws.amazon.com/cdk/v2/guide/apps.html).



## Why use the AWS CDK?

It's easier to show than to explain! Here's some CDK code that creates an Amazon ECS service with AWS Fargate launch type (this is the code we use in the [Creating an AWS Fargate service using the AWS CDK](https://docs.aws.amazon.com/cdk/v2/guide/ecs_example.html)).

**What is AWS Fargate?**

AWS Fargate is a technology that you can use with Amazon ECS to run [containers](https://aws.amazon.com/what-are-containers) without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

Fargate offers platform versions for Amazon Linux 2 and Microsoft Windows 2019 Server Full and Core editions. Unless otherwise specified, the information on this page applies to all Fargate platforms.

### TypeScript

export class MyEcsConstructStack extends Stack {

constructor(scope: App, id: string, props?: StackProps) {

super(scope, id, props);

const vpc = new ec2.Vpc(this, "MyVpc", {

maxAzs: 3 *// Default is all AZs in region*

});

const cluster = new ecs.Cluster(this, "MyCluster", {

vpc: vpc

});

*// Create a load-balanced Fargate service and make it public*

new ecs\_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService", {

cluster: cluster, *// Required*

cpu: 512, *// Default is 256*

desiredCount: 6, *// Default is 1*

taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample") },

memoryLimitMiB: 2048, *// Default is 512*

publicLoadBalancer: true *// Default is false*

});

}

}

### Python

class MyEcsConstructStack(Stack):

def \_\_init\_\_(self, scope: Construct, id: str, \*\*kwargs) -> None:

super().\_\_init\_\_(scope, id, \*\*kwargs)

vpc = ec2.Vpc(self, "MyVpc", max\_azs=3) *# default is all AZs in region*

cluster = ecs.Cluster(self, "MyCluster", vpc=vpc)

ecs\_patterns.ApplicationLoadBalancedFargateService(self, "MyFargateService",

cluster=cluster, *# Required*

cpu=512, *# Default is 256*

desired\_count=6, *# Default is 1*

task\_image\_options=ecs\_patterns.ApplicationLoadBalancedTaskImageOptions(

image=ecs.ContainerImage.from\_registry("amazon/amazon-ecs-sample")),

memory\_limit\_mib=2048, *# Default is 512*

public\_load\_balancer=True) *# Default is False*

### C#

public class MyEcsConstructStack : Stack

{

public MyEcsConstructStack(Construct scope, string id, IStackProps props=null) : base(scope, id, props)

{

var vpc = new Vpc(this, "MyVpc", new VpcProps

{

MaxAzs = 3

});

var cluster = new Cluster(this, "MyCluster", new ClusterProps

{

Vpc = vpc

});

new ApplicationLoadBalancedFargateService(this, "MyFargateService",

new ApplicationLoadBalancedFargateServiceProps

{

Cluster = cluster,

Cpu = 512,

DesiredCount = 6,

TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions

{

Image = ContainerImage.FromRegistry("amazon/amazon-ecs-sample")

},

MemoryLimitMiB = 2048,

PublicLoadBalancer = true,

});

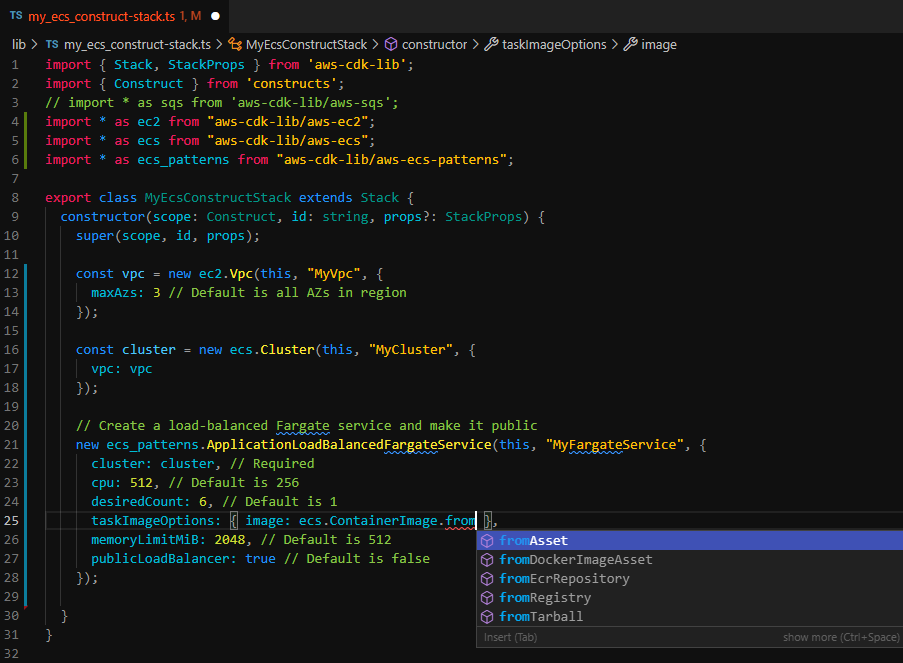
}

}

This class produces an AWS CloudFormation [template of more than 500 lines](https://github.com/awsdocs/aws-cdk-guide/blob/main/doc_source/my_ecs_construct-stack.yaml); deploying the AWS CDK app produces more than 50 resources of the following types.

* [AWS::EC2::EIP](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-eip.html)
* [AWS::EC2::InternetGateway](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-internetgateway.html)
* [AWS::EC2::NatGateway](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-natgateway.html)
* [AWS::EC2::Route](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-route.html)
* [AWS::EC2::RouteTable](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-routetable.html)
* [AWS::EC2::SecurityGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group.html)
* [AWS::EC2::Subnet](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-subnet.html)
* [AWS::EC2::SubnetRouteTableAssociation](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-subnet-route-table-assoc.html)
* [AWS::EC2::VPCGatewayAttachment](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-vpc-gateway-attachment.html)
* [AWS::EC2::VPC](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-vpc.html)
* [AWS::ECS::Cluster](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-cluster.html)
* [AWS::ECS::Service](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-service.html)
* [AWS::ECS::TaskDefinition](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-taskdefinition.html)
* [AWS::ElasticLoadBalancingV2::Listener](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-listener.html)
* [AWS::ElasticLoadBalancingV2::LoadBalancer](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-loadbalancer.html)
* [AWS::ElasticLoadBalancingV2::TargetGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-targetgroup.html)
* [AWS::IAM::Policy](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-iam-policy.html)
* [AWS::IAM::Role](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-iam-role.html)
* [AWS::Logs::LogGroup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-logs-loggroup.html)

And let's not forget... code completion within your IDE or editor!



# Docker – ECS, ECR

<https://faun.pub/what-is-amazon-ecs-and-ecr-how-does-they-work-with-an-example-4acbf9be8415>

**ECS:**

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/welcome-features.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/get-set-up-for-amazon-ecs.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/getting-started-fargate.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/getting-started-ecs-ec2-v2.html>

**ECR:**

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html>

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/get-set-up-for-amazon-ecr.html>

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/getting-started-console.html>

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/getting-started-cli.html>

**Fargate:**

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Fargate.html>

## A Complete Guide to Deploying a Node.js app on AWS with Docker

<https://jaykannaiyan.com/how-to-deploy-node-app-docker-aws>

# Kubernees – EKS

<https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html>

<https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>

<https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html>

<https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html>

<https://docs.aws.amazon.com/eks/latest/userguide/getting-started-console.html>

<https://docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html>

<https://docs.aws.amazon.com/eks/latest/userguide/horizontal-pod-autoscaler.html>

<https://docs.aws.amazon.com/eks/latest/userguide/ebs-sample-app.html>

<https://www.bmc.com/blogs/aws-ecs-vs-eks/>

# Well Architected Framework

<https://aws.amazon.com/blogs/apn/the-6-pillars-of-the-aws-well-architected-framework/>

<https://docs.aws.amazon.com/wellarchitected/latest/framework/definitions.html>

[https://aws.amazon.com/architecture/well-architected](https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc)

Every day, experts at AWS assist customers in architecting systems to take advantage of best practices in the cloud. We work with you on making architectural trade-offs as your designs evolve. As you deploy these systems into live environments, we learn how well these systems perform and the consequences of those trade-offs.

Based on what we have learned, we have created the AWS Well-Architected Framework, which provides a consistent set of best practices for customers and partners to evaluate architectures, and provides a set of questions you can use to evaluate how well an architecture is aligned to AWS best practices.

The AWS Well-Architected Framework is based on six pillars — operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability.

**Table 1. The pillars of the AWS Well-Architected Framework**

| **Name** | **Description** |
| --- | --- |
| **Operational excellence** | The ability to support development and run workloads effectively, gain insight into their operations, and to continuously improve supporting processes and procedures to deliver business value. |
| **Security** | The security pillar describes how to take advantage of cloud technologies to protect data, systems, and assets in a way that can improve your security posture. |
| **Reliability** | The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it’s expected to. This includes the ability to operate and test the workload through its total lifecycle. This paper provides in-depth, best practice guidance for implementing reliable workloads on AWS. |
| **Performance efficiency** | The ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve. |
| **Cost optimization** | The ability to run systems to deliver business value at the lowest price point. |
| **Sustainability** | The ability to continually improve sustainability impacts by reducing energy consumption and increasing efficiency across all components of a workload by maximizing the benefits from the provisioned resources and minimizing the total resources required. |

In the AWS Well-Architected Framework, we use these terms:

* A **component** is the code, configuration, and AWS Resources that together deliver against a requirement. A component is often the unit of technical ownership, and is decoupled from other components.
* The term **workload** is used to identify a set of components that together deliver business value. A workload is usually the level of detail that business and technology leaders communicate about.
* We think about **architecture** as being how components work together in a workload. How components communicate and interact is often the focus of architecture diagrams.
* **Milestones** mark key changes in your architecture as it evolves throughout the product lifecycle (design, implementation, testing, go live, and in production).
* Within an organization the **technology portfolio** is the collection of workloads that are required for the business to operate.
* The **level of effort** is categorizing the amount of time, effort, and complexity a task requires for implementation. Each organization needs to consider the size and expertise of the team and the complexity of the workload for additional context to properly categorize the level of effort for the organization.
  + **High:** The work might take multiple weeks or multiple months. This could be broken out into multiple stories, releases, and tasks.
  + **Medium:** The work might take multiple days or multiple weeks. This could be broken out into multiple releases and tasks.
  + **Low:** The work might take multiple hours or multiple days. This could be broken out into multiple tasks.

When architecting workloads, you make trade-offs between pillars based on your business context. These business decisions can drive your engineering priorities. You might optimize to improve sustainability impact and reduce cost at the expense of reliability in development environments, or, for mission-critical solutions, you might optimize reliability with increased costs and sustainability impact. In ecommerce solutions, performance can affect revenue and customer propensity to buy. Security and operational excellence are generally not traded-off against the other pillars.

## The 6 Pillars of the AWS Well-Architected Framework

Creating a software system is a lot like constructing a building. If the foundation is not solid, structural problems can undermine the integrity and function of the building.

When building technology solutions on Amazon Web Services (AWS), if you neglect the six pillars of operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability, it can become challenging to build a system that delivers on your expectations and requirements.

Incorporating these pillars into your architecture helps produce stable and efficient systems. This allows you to focus on the other aspects of design, such as functional requirements.

The [AWS Well-Architected Framework](https://aws.amazon.com/architecture/well-architected/) helps cloud architects build the most secure, high-performing, resilient, and efficient infrastructure possible for their applications. The framework provides a consistent approach for customers and [AWS Partners](https://aws.amazon.com/partners/) to evaluate architectures, and provides guidance to implement designs that scale with your application needs over time.

### 1. Operational Excellence

The Operational Excellence pillar includes the ability to support development and run workloads effectively, gain insight into their operation, and continuously improve supporting processes and procedures to delivery business value. You can find prescriptive guidance on implementation in the [Operational Excellence Pillar whitepaper](https://docs.aws.amazon.com/wellarchitected/latest/operational-excellence-pillar/welcome.html).

#### Design Principles

There are five design principles for operational excellence in the cloud:

* Perform operations as code
* Make frequent, small, reversible changes
* Refine operations procedures frequently
* Anticipate failure
* Learn from all operational failures

#### Best Practices

Operations teams need to understand their business and customer needs so they can support business outcomes. Ops creates and uses procedures to respond to operational events, and validates their effectiveness to support business needs. Ops also collects metrics that are used to measure the achievement of desired business outcomes.

Everything continues to change—your business context, business priorities, and customer needs. It’s important to design operations to support evolution over time in response to change, and to incorporate lessons learned through their performance.

### 2. Security

The Security pillar includes the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. You can find prescriptive guidance on implementation in the [Security Pillar whitepaper](https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html).

#### Design Principles

There are seven design principles for security in the cloud:

* Implement a strong identity foundation
* Enable traceability
* Apply security at all layers
* Automate security best practices
* Protect data in transit and at rest
* Keep people away from data
* Prepare for security events

#### Best Practices

Before you architect any workload, you need to put in place practices that influence security. You’ll want to control who can do what. In addition, you want to be able to identify security incidents, protect your systems and services, and maintain the confidentiality and integrity of data through data protection.

You should have a well-defined and practiced process for responding to security incidents. These tools and techniques are important because they support objectives such as preventing financial loss or complying with regulatory obligations.

The [AWS Shared Responsibility Model](https://aws.amazon.com/compliance/shared-responsibility-model/) enables organizations that adopt the cloud to achieve their security and compliance goals. Because AWS physically secures the infrastructure that supports our cloud services, as an AWS customer you can focus on using services to accomplish your goals. The AWS Cloud also provides greater access to security data and an automated approach to responding to security events.

### 3. Reliability

The Reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it’s expected to. This includes the ability to operate and test the workload through its total lifecycle. You can find prescriptive guidance on implementation in the [Reliability Pillar whitepaper](https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html).

#### Design Principles

There are five design principles for reliability in the cloud:

* Automatically recover from failure
* Test recovery procedures
* Scale horizontally to increase aggregate workload availability
* Stop guessing capacity
* Manage change in automation

#### Best Practices

Before building any system, foundational requirements that influence reliability should be in place. For example, you must have sufficient network bandwidth to your data center. These requirements are sometimes neglected (because they are beyond a single project’s scope). With AWS, however, most of the foundational requirements are already incorporated or can be addressed as needed.

The cloud is designed to be nearly limitless, so it’s the responsibility of AWS to satisfy the requirement for sufficient networking and compute capacity, leaving you free to change resource size and allocations on demand.

A reliable workload starts with upfront design decisions for both software and infrastructure. Your architecture choices will impact your workload behavior across all six AWS Well-Architected pillars. For reliability, there are specific patterns you must follow, such as loosely coupled dependencies, graceful degradation, and limiting retries.

Changes to your workload or its environment must be anticipated and accommodated to achieve reliable operation of the workload. Changes include those imposed on your workload, like a spikes in demand, as well as those from within such as feature deployments and security patches.

Low-level hardware component failures are something to be dealt with every day in an on-premises data center. In the cloud, however, these are often abstracted away. Regardless of your cloud provider, there is the potential for failures to impact your workload. You must therefore [take steps to implement resiliency](https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/shared-responsibility-model-for-resiliency.html) in your workload, such as fault isolation, automated failover to healthy resources, and a disaster recovery strategy.

### 4. Performance Efficiency

The Performance Efficiency pillar includes the ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve. You can find prescriptive guidance on implementation in the [Performance Efficiency Pillar whitepaper](https://docs.aws.amazon.com/wellarchitected/latest/performance-efficiency-pillar/welcome.html).

#### Design Principles

There are five design principles for performance efficiency in the cloud:

* Democratize advanced technologies
* Go global in minutes
* Use serverless architectures
* Experiment more often
* Consider mechanical sympathy

#### Best Practices

Take a data-driven approach to building a high-performance architecture. Gather data on all aspects of the architecture, from the high-level design to the selection and configuration of resource types.

Reviewing your choices on a regular basis ensures you are taking advantage of the continually evolving AWS Cloud. Monitoring ensures you are aware of any deviance from expected performance. Make trade-offs in your architecture to improve performance, such as using compression or caching, or relaxing consistency requirements

The optimal solution for a particular workload varies, and solutions often combine multiple approaches. AWS Well-Architected workloads use multiple solutions and enable different features to improve performance

### 5. Cost Optimization

The Cost Optimization pillar includes the ability to run systems to deliver business value at the lowest price point. You can find prescriptive guidance on implementation in the [Cost Optimization Pillar whitepaper](https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/welcome.html).

#### Design Principles

There are five design principles for cost optimization in the cloud:

* Implement cloud financial management
* Adopt a consumption model
* Measure overall efficiency
* Stop spending money on undifferentiated heavy lifting
* Analyze and attribute expenditure

#### Best Practices

As with the other pillars, there are trade-offs to consider. For example, do you want to optimize for speed to market or for cost? In some cases, it’s best to optimize for speed—going to market quickly, shipping new features, or simply meeting a deadline—rather than investing in up-front cost optimization.

Design decisions are sometimes directed by haste rather than data, and as the temptation always exists to overcompensate rather than spend time benchmarking for the most cost-optimal deployment. This might lead to over-provisioned and under-optimized deployments.

Using the appropriate services, resources, and configurations for your workloads is key to cost savings.

### 6. Sustainability

The discipline of sustainability addresses the long-term environmental, economic, and societal impact of your business activities. You can find prescriptive guidance on implementation in the [Sustainability Pillar whitepaper](https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html).

#### Design Principles

There are six design principles for sustainability in the cloud:

* Understand your impact
* Establish sustainability goals
* Maximize utilization
* Anticipate and adopt new, more efficient hardware and software offerings
* Use managed services
* Reduce the downstream impact of your cloud workloads

#### Best Practices

Choose AWS Regions where you will implement workloads based on your business requirements and sustainability goals.

User behavior patterns can help you identify improvements to meet sustainability goals. For example, scale infrastructure down when not needed, position resources to limit the network required for users to consume them, and remove unused assets.

Implement software and architecture patterns to perform load smoothing and maintain consistent high utilization of deployed resources. Understand the performance of your workload components, and optimize the components that consume the most resources.

Analyze data patterns to implement data management practices that reduce the provisioned storage required to support your workload. Use lifecycle capabilities to move data to more efficient, less performant storage when requirements decrease, and delete data that’s no longer required.

Analyze hardware patterns to identify opportunities that reduce workload sustainability impacts by minimizing the amount of hardware needed to provision and deploy. Select the most efficient hardware for your individual workload.

In your development and deployment process, identify opportunities to reduce your sustainability impact by making changes, such as updating systems to gain performance efficiencies and manage sustainability impacts. Use automation to manage the lifecycle of your development and test environments, and use managed device farms for testing.

# Cloud Best Practices

<https://www.cloud.northwestern.edu/resources/cloud-best-practices/>

<https://www.buchanan.com/cloud-computing-best-practices/>

The Northwestern Cloud Community of Practice recommends these best practices for effective operation and management of public cloud services.

* Perform operations as code
* Use version control for configuration as well as code
* Automate security practices and controls
* Learn from operational failures and share learnings across the organization

## Perform operations as code

The API-driven, programmable nature of cloud environments allows practices traditionally used in software engineering to be applied to cloud infrastructure.

Operational tasks and procedures should be automated to the maximum extent possible using a combination of scripting environments like Bash and Powershell, along with configuration management and automation tools such as Terraform and Ansible.

By performing operations as code, you will limit errors, increase productivity, and free up your time to work on higher level projects.

## Use version control for configuration as well as code

All configuration for your cloud resources can be managed with the same tools and discipline as your application code by using a version control system. This will also enable continuous delivery practices that greatly enhance efficiency, visibility, consistency, and security.

The Cloud Community of Practice recommends the adoption of git as a version control system, along with a hosted git provider such as [GitHub](https://github.com/). When git is used in combination with a continuous integration / continuous deployment (CI/CD) tool such as [GitHub Actions](https://github.com/features/actions) or [Jenkins](https://jenkins.io/), configuration changes can be automatically tested and deployed.

## Architect for security

Implementing basic security practices everywhere can protect you from misconfigurations, mistakes, and successful attacks.

Control access to cloud resources by implementing role-based access control using IAM for AWS and [Azure Active Directory](https://azure.microsoft.com/services/active-directory/). Make sure your roles have the minimum permissions they need to perform their functions and enable Multi-Factor Authentication (MFA) to further protect access.

Ensure storage accounts do not allow public access and enable encryption across all storage services wherever possible. Additionally, enabling object versioning in cloud storage buckets can protect against accidental data deletion.

Use the cloud provider’s security services to enable automatic, programmatic remediation of security misconfigurations (AWS Config, Azure Policy), get visibility across cloud services (AWS Security Hub, Azure Security Center), and protect web services from attack (AWS Shield, Web Application Firewall).

## Learn from operational failures and share learnings across the organization

Failures and incidents will occur no matter how much you work to prevent them, so lay the groundwork to quickly identify incidents and recover as quickly as possible. Write and share a blameless post-mortem of each incident to help the organization learn and become more resilient.

Having regular “Game Days” is good practice for teams to analyze failures and identify lessons learned. Create a strategy document of lessons learned and revisit the documentation after each game day. Share what is learned across teams and with the Cloud Community of Practice.

Setting up the isolated test environment and adopting the principles of [Chaos Engineering](https://en.wikipedia.org/wiki/Chaos_engineering) can help you to go through the process of identifying failures before they become outages.

## **Top 10 Practices for Managing the Cloud**

There are several cloud computing best practices that business owners need to follow in 2023 for effective cloud adoption that provides long-term benefits.

Let’s take a look at the top 10 cloud computing best practices.

### 1. Create a Cloud Center of Excellence (CCoE) for Cloud Application Management

Business owners who want to ensure success in their cloud adoption process must follow the proper structure and have the right skills for optimal execution. Setting up a cloud center of excellence (CCoE) is the best way to ensure these requirements are met.

This central governance committee oversees cloud computing practices and is responsible for essential tasks such as setting the cloud policy and focusing on managing risks and improving outcomes.

A good CCoE includes finance experts and IT who can properly carry out their consultative role regarding the organization’s core IT infrastructure.

### 2. Assess Business Goals and Understand the Benefits

Before diving into cloud adoption for your business, you must first develop clear goals on what you want to gain for your organization by moving to the cloud.

Business owners should identify and list concrete reasons why moving to the cloud is critical for their business growth, these may include benefits such as:

* Reduced risk
* Stronger security tools
* Long-term IT operating cost savings
* Higher availability
* Elastic capacity
* Cost reduction
* Simplify my IT management
* Eliminate hardware
* Modernize our environment
* Upgrade to cloud application, e.g., Salesforce or Dynamics

When an organization takes a cloud-first approach, it must shift all its data to the cloud to benefit optimally. An end-to-end assessment that considers opportunities, costs, and risks will ascertain that business owners thoroughly understand their cloud adoption implications and goals.

### 3. Select the Best Model

There are various types of cloud computing organizations can use.

Whether an organization opts for private, hybrid, or public cloud computing often depends on the available budget and long-term business goals.

*For instance, businesses looking for cloud adoption with no maintenance costs may opt for public cloud computing. However, if cloud security is a concern and companies want to focus on protecting sensitive data, then private cloud computing is a better option. Hybrid cloud computing is a good fit for businesses seeking flexibility and policy-driven deployment.*

Cost is of considerable concern for organizations migrating to the cloud.

Organizations need to compare the cost difference between supported cloud models. Additionally, they need to sensical coordinate them with other business aspects *(to ensure the best one is selected accordingly)*.

### 4. Understand the Distinct Areas of Cloud Adoption

It is vital to differentiate your cloud adoption to reduce risks and ensure effective cloud implementation. Business owners not only have to understand the different models of cloud computing, but also the different areas of cloud adoption.

These include software as a service (SaaS), Cloud infrastructure platforms (CIPS),  or possibly migrating legacy applications or outdated systems no longer supported into a cloud environment.

Each approach should be assessed and implemented accordingly, but the roadmap should be considered so your goals are in line with your steps to adoption

### 5. Establish Governance for Managing Cloud Services

When companies are able to establish good governance for the cloud, they eliminate waste and prevent unmanaged growth.

By focusing on implementing proper governance during cloud adoption, IT processes are sufficiently optimized, which improves the chance of business growth and success.

### 6. Continuously Optimize Processes

Once the cloud adoption process begins, business owners have to continuously revisit and remodel their processes.

To ensure continued and optimal success in cloud adoption, organizations have to keep updating essential matters such as value measurements, budgets, and CCoE processes.

Cloud learning and education should also be offered across the organization so there is continuous IT infrastructure improvement.

### 7. Ensure Data is Actionable

Organizations need to keep in mind that any data provided by the cloud is raw and should only be implemented based on the context. Business owners need to evaluate any recommendations from cloud service providers and ask the “why” behind the recommendation.

Most of the time it will be for one of the benefits outlined in step 2 above, but businesses should take different scenarios into consideration to ensure they are making the best possible moves.

### 8. Prioritize Communication

One of the main keys to ensuring successful cloud adoption is to ensure there is good communication across the entire organization.

Important factors such as the main benefits of cloud migration as well as cost implications should be discussed properly with the appropriate departments so no avoidable complications arise later down the road.

### 9. Take Advantage of Automation

Cloud integration is known for its intricate nature and can take up a lot of time to carry out. To ensure that operating costs are reduced by as much as possible, organizations should try their best to automate certain cloud-based tasks such as configuration and management.

Automating tasks will give business owners the time they need to focus on perfecting their business plan and growing their business. Most security issues within the cloud are caused by misconfigurations, so this point cannot be emphasized enough.

Using automation can take time and money to build, but especially in compliance situations, it’s worth the investment to take out human error and utilize your team for spot checks.

### 10. Get Professional Assistance

Implementing cloud adoption is a complex process that needs to be undertaken with the utmost care. Organizations should ensure they only carry out cloud migration with the help of a cloud professional who has sufficient business and IT experience.

Even a small error during the migration can lead to serious losses and the potential risk of a data breach so it is always a good idea to get in touch with a Cloud Service Provider (CSP) who specializes in cloud migration services.