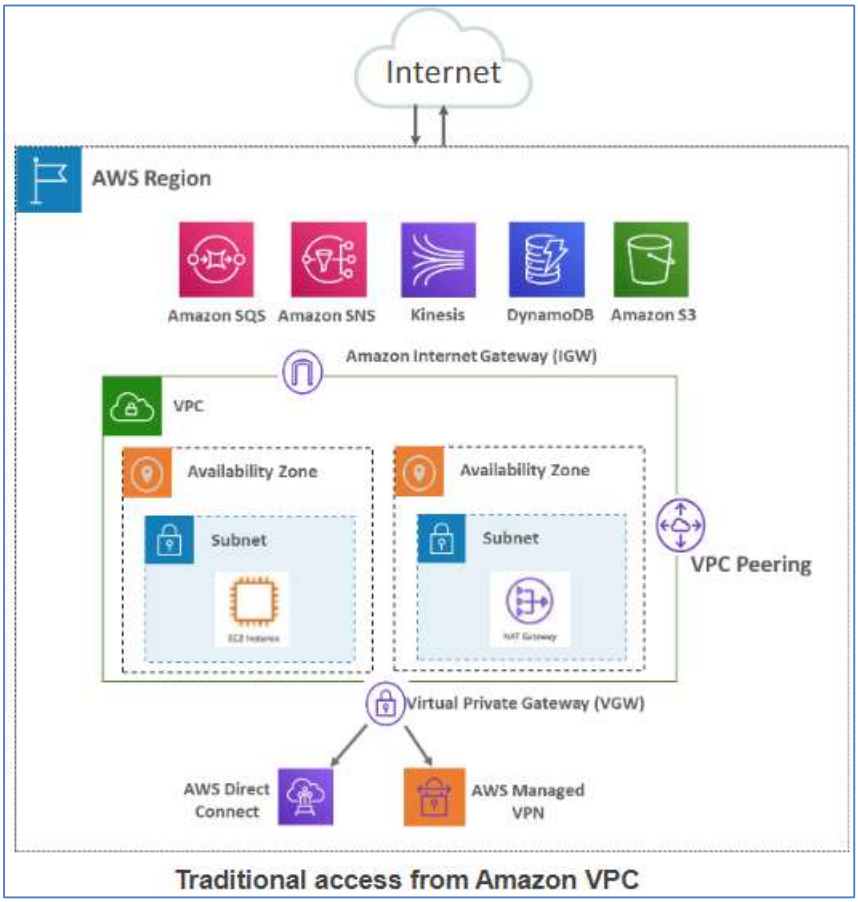
Contents

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.

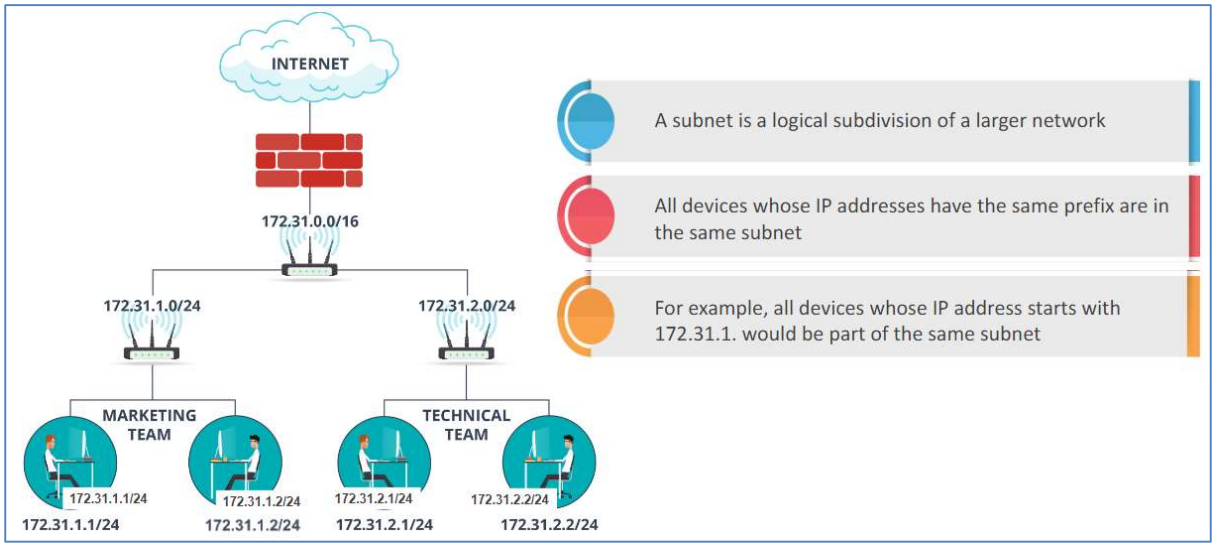
As one of AWS's foundational services, Amazon VPC makes it easy to customize your VPC's network configuration. You can create a public-facing subnet for your web servers that have access to the internet. It also lets you place your backend systems, such as databases or application servers, in a private-facing 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

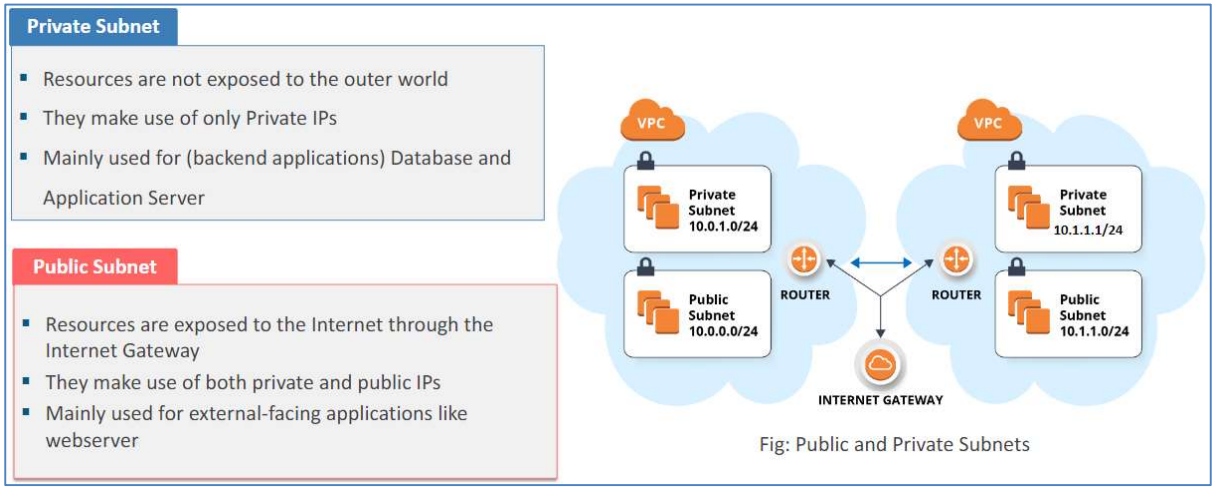


Fig: Public and Private Subnets

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.

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

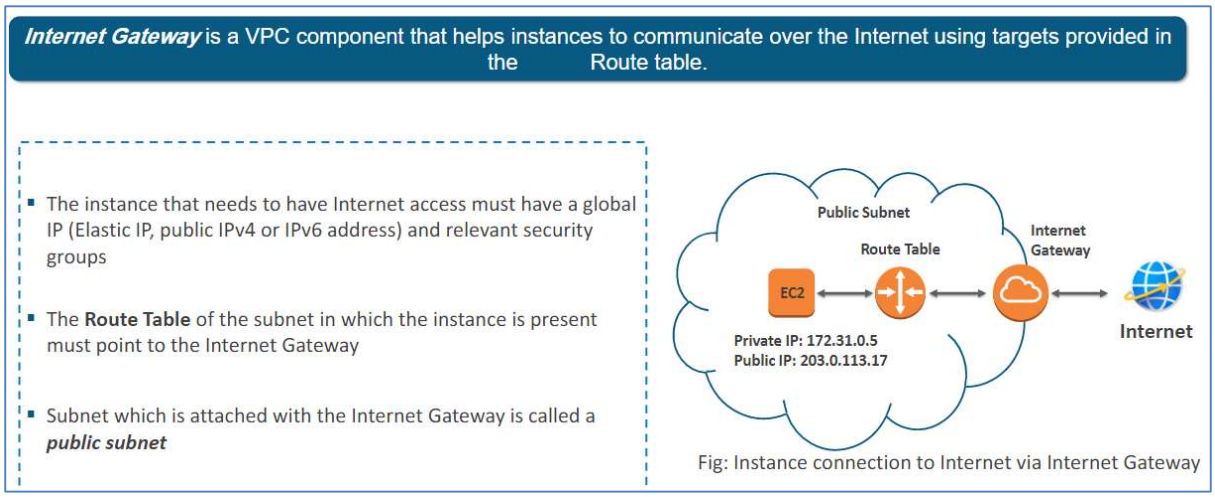
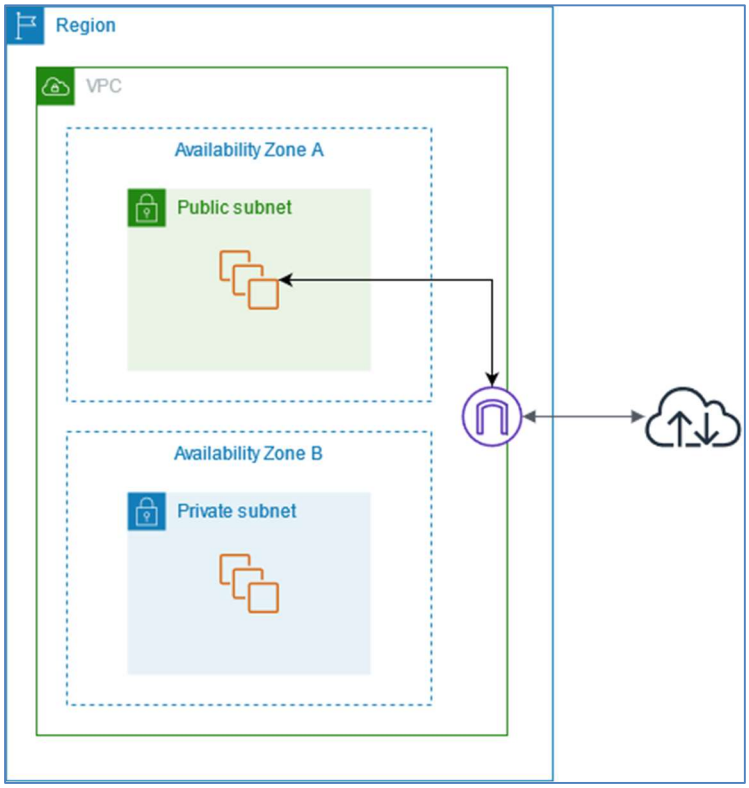


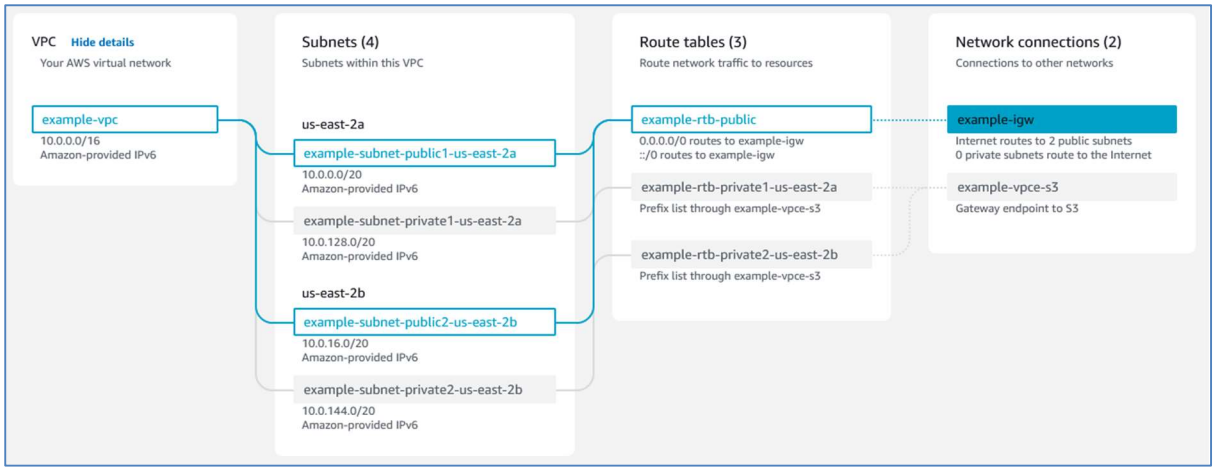
Fig: Instance connection to Internet via Internet Gateway



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).



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.

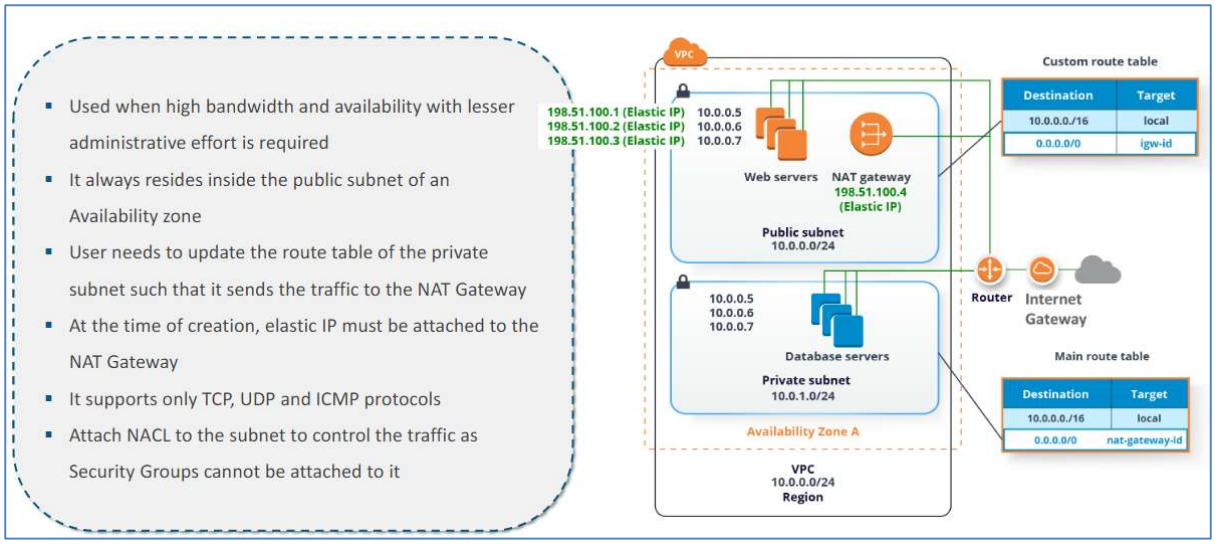
When you create a NAT gateway, you specify one of the following connectivity types:

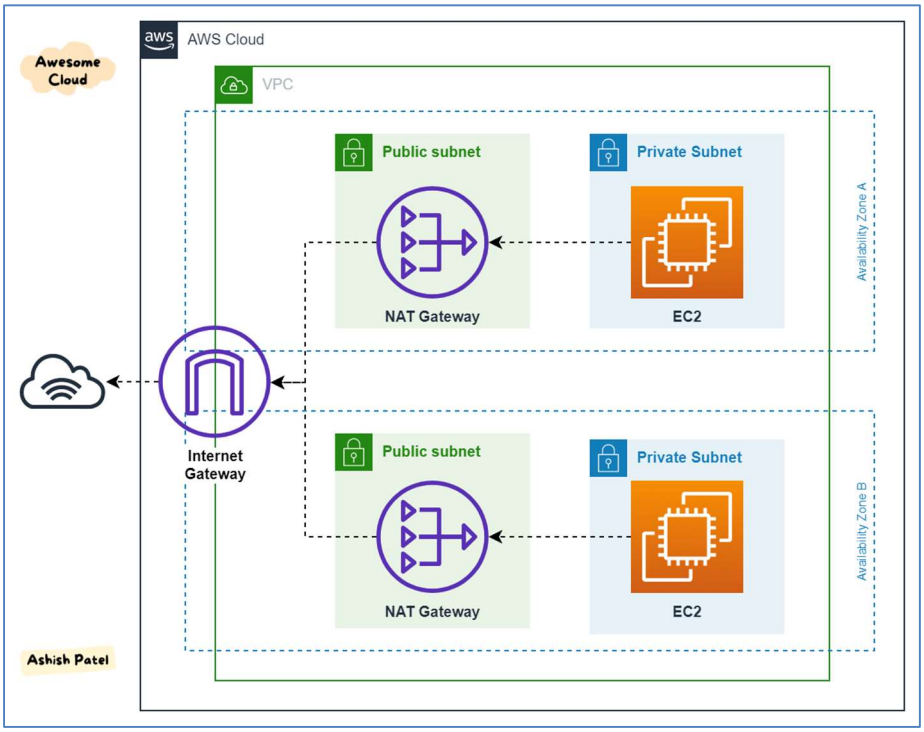
- **Public** – (Default) Instances in private subnets can connect to the internet through a public NAT gateway, but cannot receive unsolicited inbound connections from the internet. You create a public NAT gateway in a public subnet and must associate an elastic IP address with the NAT gateway at creation. 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

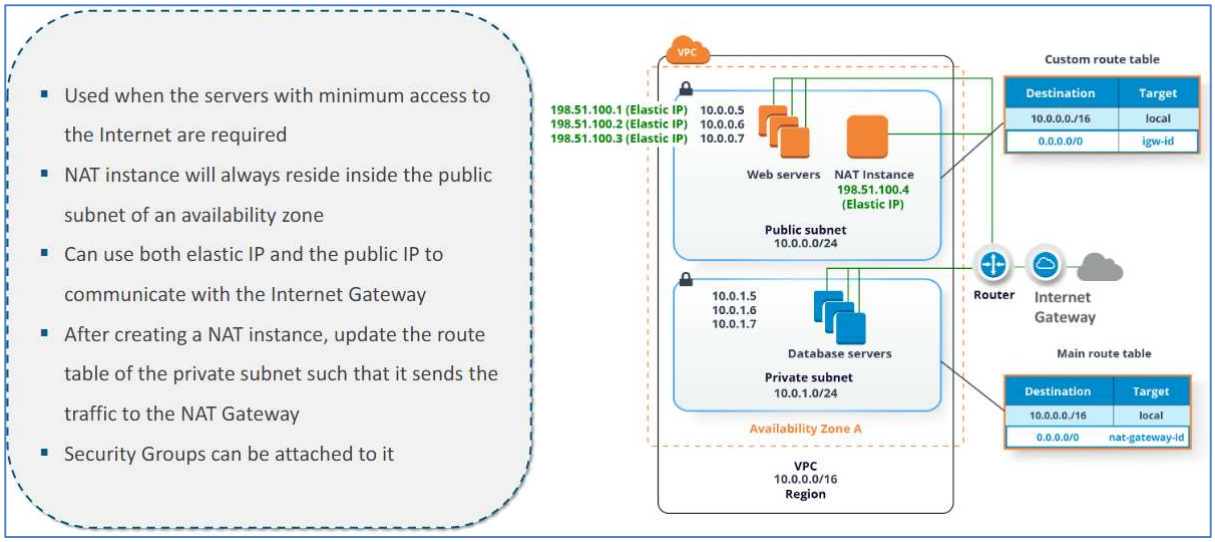


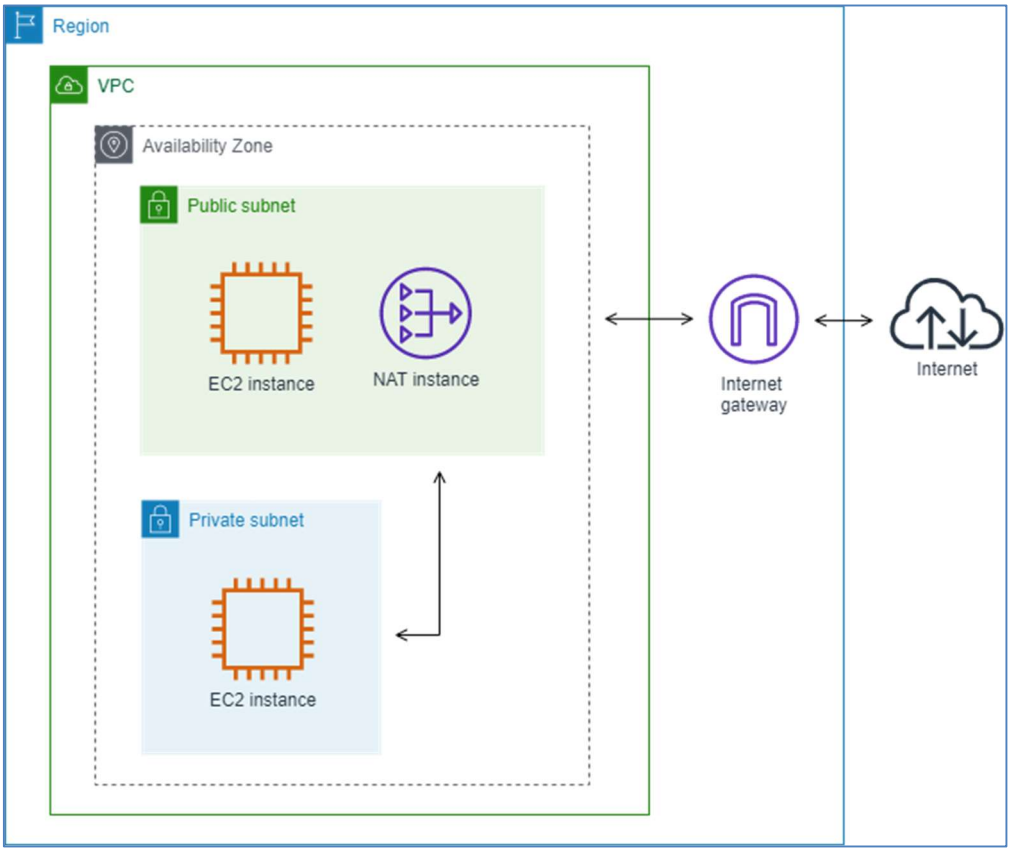


NAT Instance

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 in the *Amazon EC2 User Guide for Linux Instances*.





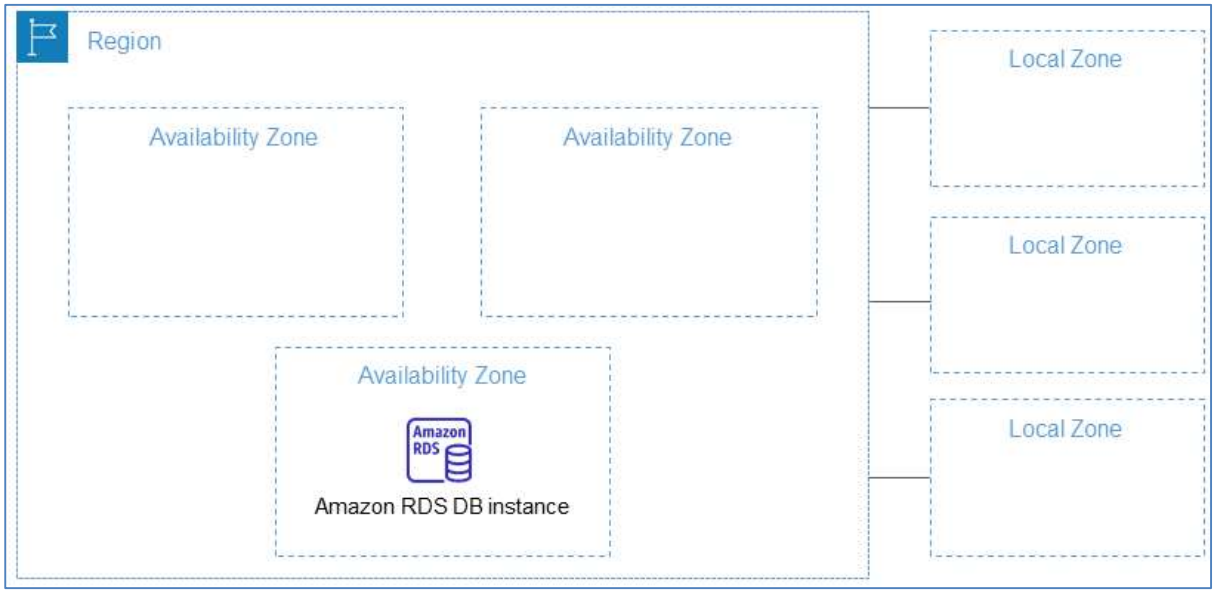
Regions, Availability Zones and Local Zones

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.



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` 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, 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)?

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 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.

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 and Modifying an Amazon RDS DB instance.

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.

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 in the *Amazon EC2 User Guide for Linux Instances.*

2. Create a subnet in the Local Zone.

For more information, see Creating a subnet in your VPC in the *Amazon VPC User Guide.*

3. 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.

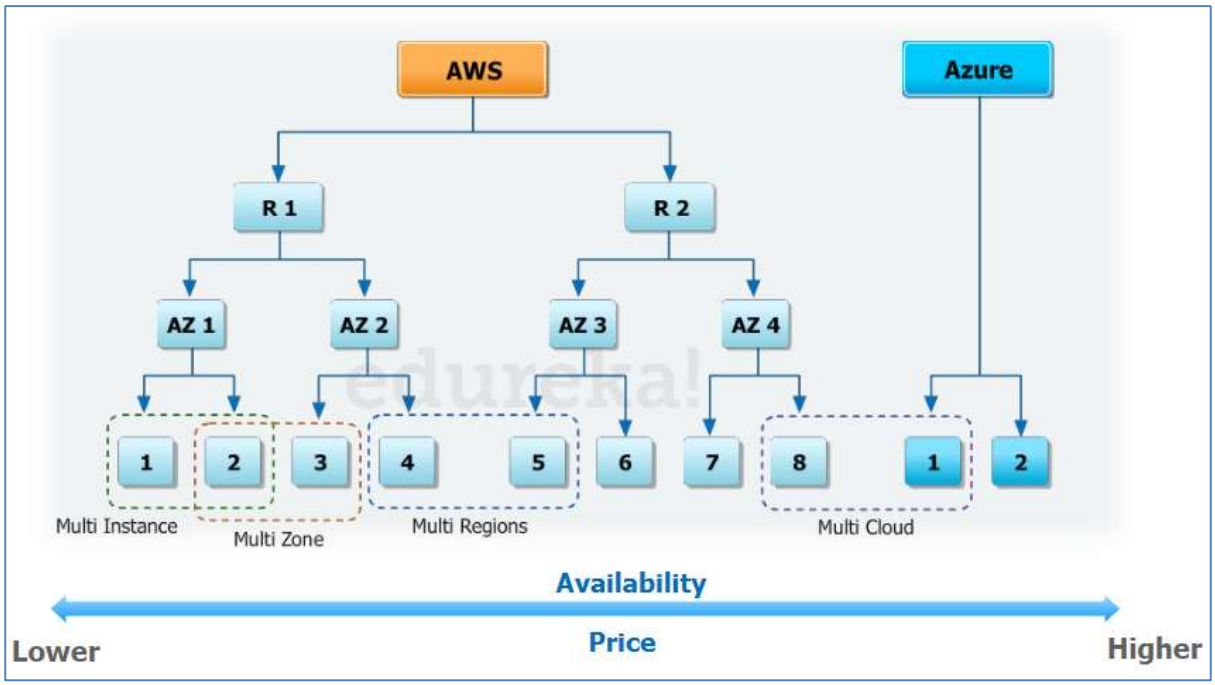
   For more information, see Creating a DB instance in a VPC.

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

   For more information, see Creating an Amazon RDS DB instance.

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
    a. Server manager
    b. Add roles and features
    c. Role-based or feature-based installation
    d. Select the server
    e. Select Web Server (IIS)
    f. Features -> .NET Framework 4.8…
    g. 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

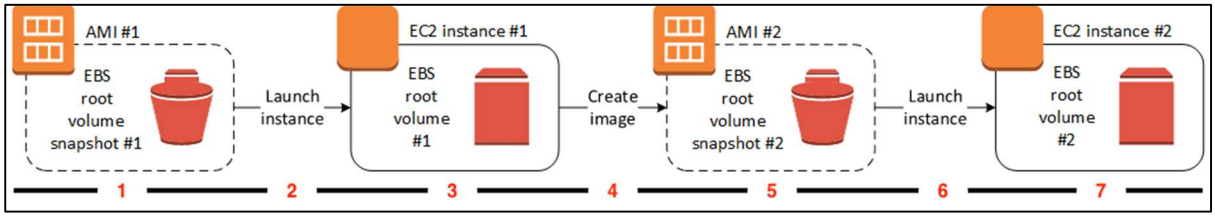
Create Custom AMI

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.



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 or VM Import/Export, 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 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.

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.

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.

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.

Steps:

1. Create a new Linux instance
2. Select the instance -> Actions -> Images and Templates -> Create image
    a. For **Image name**, enter a unique name for the image, up to 127 characters.
    b. For **Image description**, enter an optional description of the image, up to 255 characters.
    c. 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.

    d. **Instance volumes** – You can modify the root volume, and add additional Amazon EBS and instance store volumes, as follows:
        i. 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.
        ii. 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.
        iii. To add an instance store volume, see Add instance store volumes to an 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.
    e. **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.

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

3. To view the status of your AMI while it is being created:
   a. In the navigation pane, choose **AMIs**.
   b. 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.
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
   ```

3. 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
   ```

4. 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.
```

5. Start the Apache web server.

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

6. 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
```

7. 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.
    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
    b. In the left navigator, choose **Instances**, and select your instance.
    c. 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.

   d. 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, add a new inbound security rule with the following values:
      - **Type**: HTTP
      - **Protocol**: TCP
      - **Port Range**: 80
      - **Source**: Custom

8. 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.

> **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
```

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

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

   b. 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
```

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

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

4. 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 {} \;
```

5. 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.

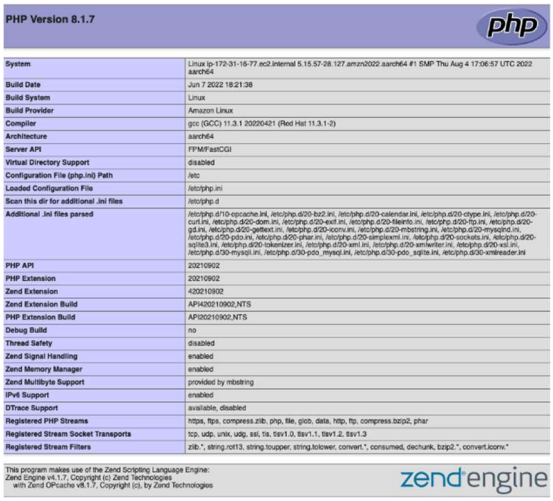
```
[ec2-user ~]$ echo "<?php phpinfo(); ?>" >
/var/www/html/phpinfo.php
```

   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.

2. 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.

3. 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
```

2. Run mysql_secure_installation.

```
[ec2-user ~]$ sudo mysql_secure_installation
```

   a. When prompted, type a password for the root account.
      i. Type the current root password. By default, the root account does not have a password set. Press Enter.
      ii. 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.

   b. Type Y to remove the anonymous user accounts.
   c. Type Y to disable the remote root login.
   d. Type Y to remove the test database.
   e. Type Y to reload the privilege tables and save your changes.

3. (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
```

4. (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 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. For general information about securing a web server on an EC2 instance, see Tutorial: Configure SSL/TLS on Amazon Linux 2023.

To install phpMyAdmin

1. Install the required dependencies.

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

2. Restart Apache.

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

3. Restart `php-fpm`.

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

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

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

5. 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
```

6. 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
```

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

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

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

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

9. 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:

10. 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:

    a. 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.
    b. Save the file as `config.inc.php` in the phpMyAdmin directory that contains `index.php`.
    c. Refer to post-file creation instructions in the 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. When a role serves a specialized purpose for a service, it can be categorized as a service role for EC2 instances, or a service-linked role. See the AWS documentation 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.

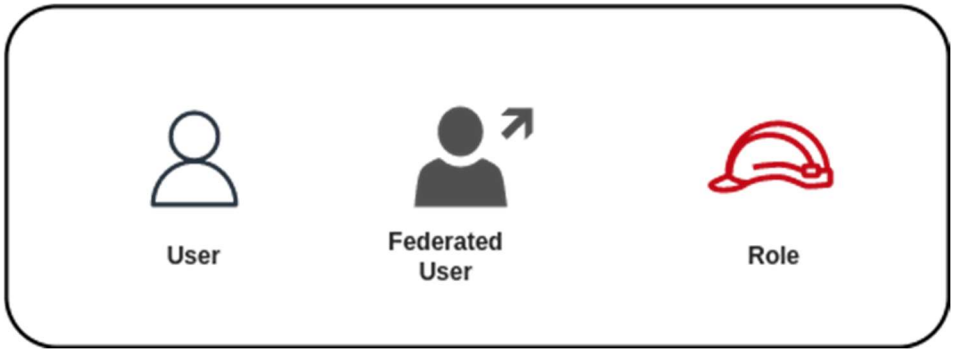
Identities: Roles, Users and Federated Identities

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

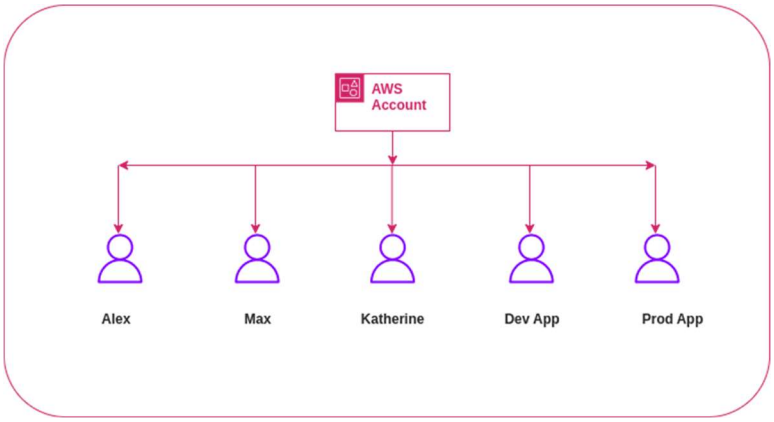


AWS Users

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

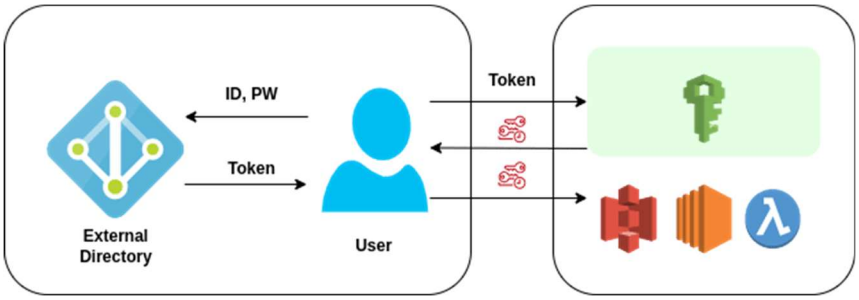
We can also *generate access keys* that are tied to a particular user that can be used with the AWS CLI or SDKs.

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.

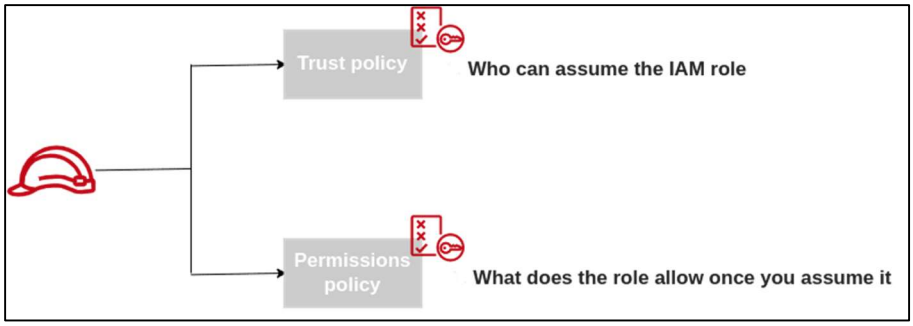
In fact, AWS recommends using temporary IAM roles over IAM users.

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.

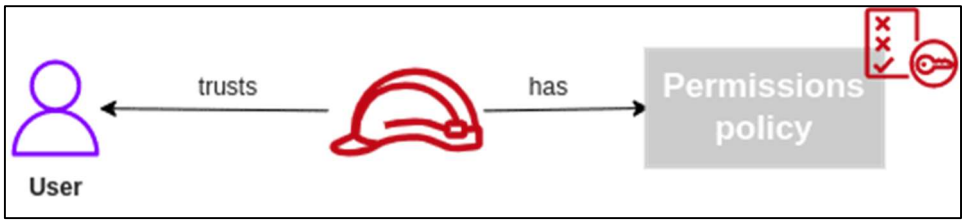
With the fundamentals of an IAM role established, we can proceed to create our first IAM role.

Creating Our First Role

Creating our first role

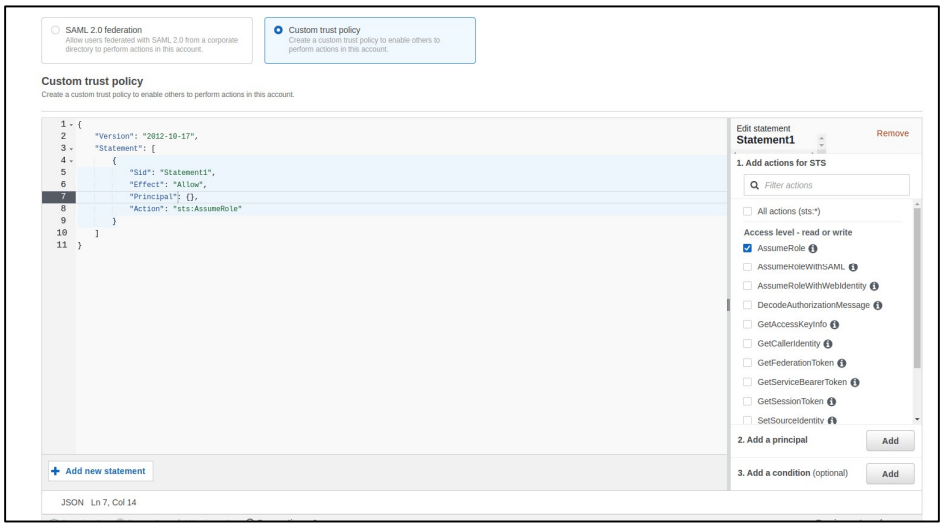
Pre-requisite: An AWS account with an IAM admin user.

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.



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.

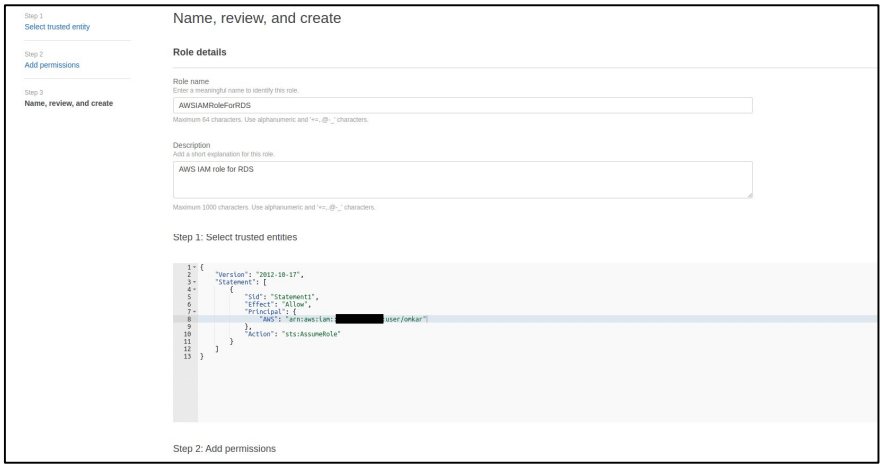
6. Upon selection of the “Custom trust policy”, AWS automatically generates a JSON policy with an empty principal field. The principal field specifies who can assume this role. If we keep it empty then this policy cannot be assumed by any principal.



7. We will add the Amazon Resource Name (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.
8. 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"
 }
]
}
```

9. Once you have reviewed the trust policy, click on the next button to move on to the next page.
10. 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.
11. **Remember that AWS denies everything by default.** We are only granting the RDS access to this role by attaching this policy.
12. Leave all the other settings unchanged and click on next.
13. 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.



14. Name the role as `AWSIAMRoleForRDS` and provide the description as “AWS IAM role for RDS”
15. 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.

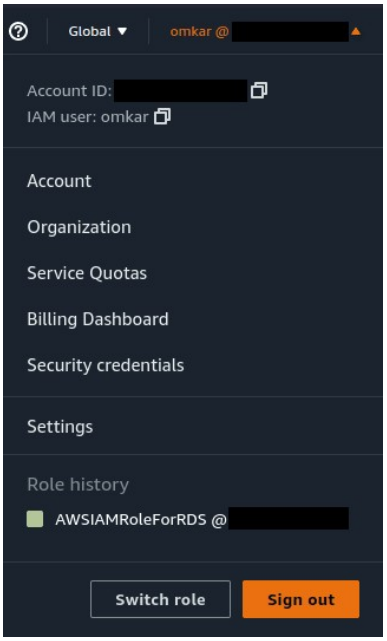
Assuming IAM Roles

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.



2. 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.
3. 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.

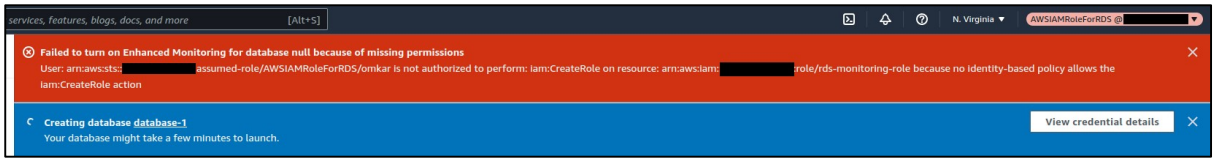
Using an IAM role to create resources

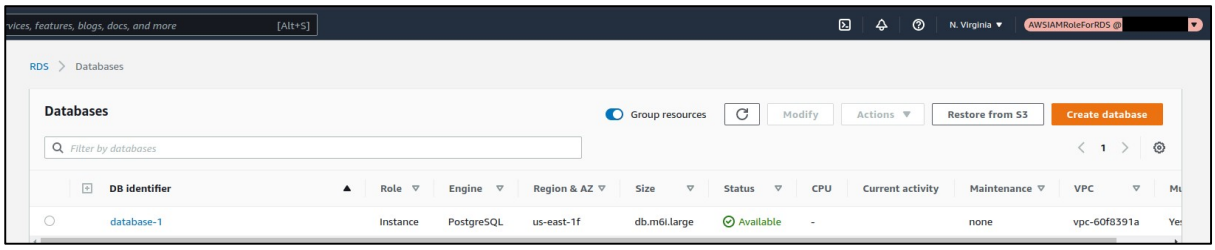
We granted our IAM role complete access to AWS RDS. This means that it has read, written as well as created permission for the RDS service. Before proceeding, please make sure that you have already assumed the AWSRDSIAMRole role

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 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.





3. 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 `survey.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 ]
      }
   ]
}
```

12. Type the above bucket policy and save it.
13. Create an index.html page on your local system
14. 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>
```

15. Upload index.html to your bucket
16. 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 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) 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.

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.

```
{
   "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.

```
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.

```
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.

```
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. The `aws:SourceIp` IPv4 values use standard CIDR notation. For more information, see IAM JSON Policy Elements Reference 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
    a. Associate to VPC
    b. Associate public subnet
    c. Add routes: 0.0.0.0/0 and select our IGW.
6. Create an EC2 instance – Window – Pubic subnet
    a. Windows
    b. Key pair
    c. Our VPC
    d. Public subnet
    e. Auto-assign public IP - Yes
    f. Security Group Inbound rules:
        i. RDP - TCP – 3389 – Anywhere-IPv4
        ii. HTTP - HTTP – 80– Anywhere-IPv4
        iii. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    g. Launch
7. Connect to instance
    a. Download RDP file
    b. Get password from .pem file (of key pair)
    c. Username: Administrator
    d. Browser check
8. Create NAT Gateway (NGW)
    a. Select public subnet
    b. Connectivity type: Public
    c. Attach Elastic IP
9. Create a Route Table for private subnet
    a. Associate to VPC
    b. Associate private subnet
    c. Add routes: 0.0.0.0/0 and select our NGW.
10. Create an EC2 instance – Window – Private subnet
    a. Windows
    b. Key pair
    c. Our VPC
    d. Private subnet
    e. Auto-assign public IP - No
    f. Security Group Inbound rules: Use same SG created with public instance
        i. RDP - TCP – 3389 – Anywhere-IPv4
        ii. HTTP - HTTP – 80– Anywhere-IPv4
        iii. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    g. Launch

11. Connect to instance using private IP
    a. Download RDP file
    b. Get password from .pem file (of key pair)
    c. Username: Administrator
    d. 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
    a. Associate to VPC (US)
    b. Associate public subnet
    c. Add routes: 0.0.0.0/0 and select our US-IGW.
7. Create an EC2 instance – Window – Pubic subnet
    a. Windows
    b. Create New Key pair
    c. Our VPC (US)
    d. Public subnet
    e. Auto-assign public IP - Yes
    f. Security Group Inbound rules:
        i. RDP - TCP – 3389 – Anywhere-IPv4
        ii. HTTP - HTTP – 80– Anywhere-IPv4
        iii. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    g. Launch
8. Connect to instance
    a. Download RDP file
    b. Get password from .pem file (of key pair)
    c. Username: Administrator
    d. Browser check
9. Create NAT Gateway (NGW) – Name: US-NGW
    a. Select public subnet
    b. Connectivity type: Public
    c. Attach Elastic IP
10. Create a Route Table for private subnet
    a. Associate to VPC (US)
    b. Associate private subnet
    c. Add routes: 0.0.0.0/0 and select our US-NGW.
11. Create an EC2 instance – Window – Private subnet
    a. Windows
    b. Use same Key pair
    c. Our VPC (US)
    d. Private subnet
    e. Auto-assign public IP - No
    f. Security Group Inbound rules: Use same SG created with public instance
        i. RDP - TCP – 3389 – Anywhere-IPv4

        ii. HTTP - HTTP – 80– Anywhere-IPv4
        iii. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    g. Launch
12. Connect to instance using private IP
    a. Download RDP file
    b. Get password from .pem file (of key pair)
    c. Username: Administrator
    d. 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
    a. Associate to VPC (EU)
    b. Associate public subnet
    c. Add routes: 0.0.0.0/0 and select our EU-IGW.
7. Create an EC2 instance – Window – Pubic subnet
    a. Windows
    b. Create new Key pair
    c. Our VPC (EU)
    d. Public subnet
    e. Auto-assign public IP - Yes
    f. Security Group Inbound rules:
        i. RDP - TCP – 3389 – Anywhere-IPv4
        ii. HTTP - HTTP – 80– Anywhere-IPv4
        iii. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
    g. Launch
8. Connect to instance
    a. Download RDP file
    b. Get password from .pem file (of key pair)
    c. Username: Administrator
    d. Browser check
9. Create NAT Gateway (NGW) – Name: EU-NGW
    a. Select public subnet
    b. Connectivity type: Public
    c. Attach Elastic IP
10. Create a Route Table for private subnet
    a. Associate to VPC (EU)
    b. Associate private subnet
    c. Add routes: 0.0.0.0/0 and select our EU-NGW.
11. Create an EC2 instance – Window – Private subnet
    a. Windows

b. Use same Key pair
c. Our VPC (EU)
d. Private subnet
e. Auto-assign public IP - No
f. Security Group Inbound rules: Use same SG created with public instance
    i. RDP - TCP – 3389 – Anywhere-IPv4
    ii. HTTP - HTTP – 80– Anywhere-IPv4
    iii. All ICMP Ipv4 - ICMP– All– Anywhere-IPv4
g. Launch

12. Connect to instance using private IP
    a. Download RDP file
    b. Get password from .pem file (of key pair)
    c. Username: Administrator
    d. 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	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	
IPv4 CIDR blocks per VPC	5	Yes (up to 50)	This primary CIDR block and all secondary CIDR blocks count toward this quota.
IPv6 CIDR blocks per VPC	5	Yes (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 Technical Guide.

Elastic IP addresses

Name	Default	Adjustable	Comments
Elastic IP addresses per Region	5	Yes	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 in the *AWS General Reference*.	

Gateways

Name	Default	Adjustable	Comments
Egress-only internet gateways per Region	5	Yes	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	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	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	You can associate one network ACL to one or more subnets in a VPC.
Rules per network ACL	20	Yes	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.
Network interfaces per Region	5,000	Yes	This quota applies to individual AWS account VPCs and shared VPCs.

Route tables

Name	Default	Adjustable	Comments
Route tables per VPC	200	Yes	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	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	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	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. 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 (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 (up to 125)	If you increase this quota, you should increase the number of entries per route table accordingly.

Name	Default	Adjustable	Comments
Outstanding VPC peering connection requests	25	Yes	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 in the *Amazon VPC Peering Guide*.

VPC endpoints

Name	Default	Adjustable	Comments
Gateway VPC endpoints per Region	20	Yes	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 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 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.

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. 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.

5. 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**.
6. (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.
7. (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).
8. 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.
9. From the **Allow/Deny** list, select **ALLOW** to allow the specified traffic or **DENY** to deny the specified traffic.
10. (Optional) To add another rule, choose **Add another rule**, and repeat steps 4 to 9 as required.
11. 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)

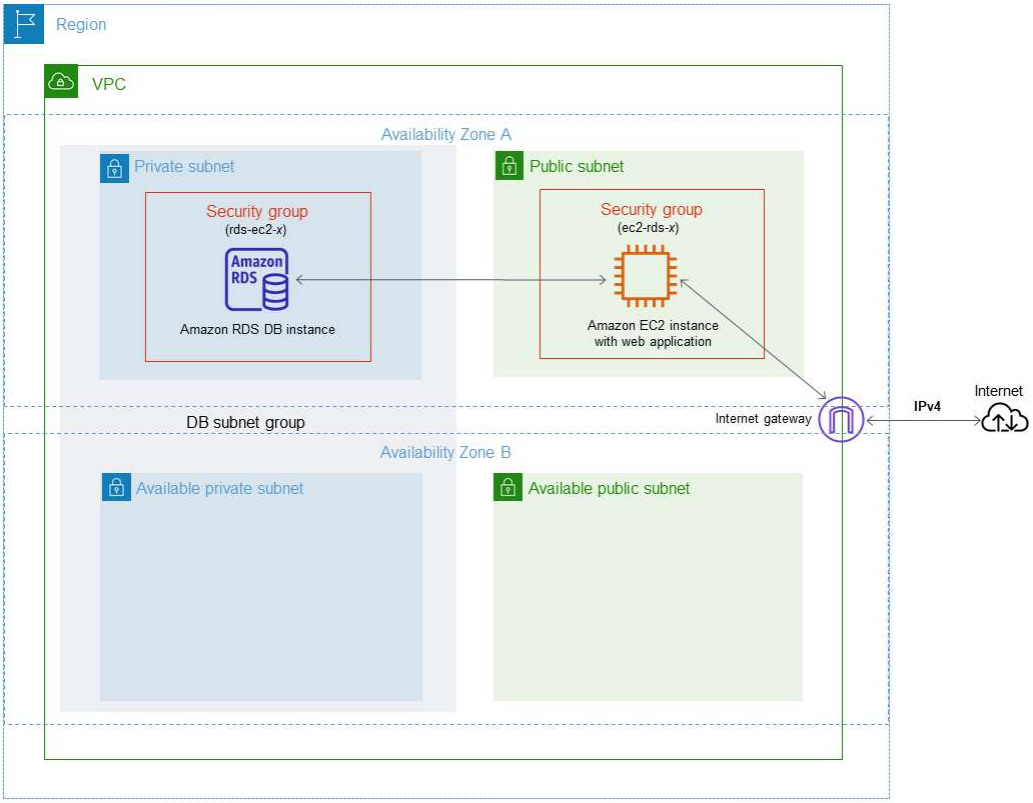
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateVPC.html

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.

6. 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.
   a. 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. 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.

b. In the **Inbound rules** section, choose **Add rule**.
c. 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**.

d. Choose **Add rule**.
e. 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**

5. 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.
    a. In the **Inbound rules** section, choose **Add rule**.
    b. 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.
    a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
    b. Choose **VPC Dashboard**, and then choose **Subnets**.
    c. 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.

2. 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.

3. In the navigation pane, choose **Subnet groups**.
4. Choose **Create DB subnet group**.
5. 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`***)
6. In the **Add subnets** section, choose the **Availability Zones and Subnets**.
7. 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.
8. 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.
    a. Under Name and tags, for Name, enter **`ec2-database-connect`**.
    b. 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.
    c. Under **Instance type**, choose **t2.micro**.
    d. 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 in the *Amazon EC2 User Guide for Linux Instances*.

    e. 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).
       - 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. 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.

f. Leave the default values for the remaining sections.

g. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-03a6ad47e97ba9dc5`.
6. Choose **View all instances** to find your instance.
7. Wait until **Instance state** for your instance is **Running** before continuing.
8. Complete Create a DB instance.

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.
10. In the **Instance configuration** section, set these values:
    - Burstable classes (includes t classes)
    - db.t3.micro
11. In the **Storage** section, keep the defaults.

12. 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, such as tutorial-ec2-instance-web-server.
13. In the Database authentication section, make sure Password authentication is selected.
14. Open the Additional configuration section, and enter `sample` for Initial database name. Keep the default settings for the other options.
15. To create your MySQL DB instance, choose Create database.

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

16. Wait for the **Status** of your new DB instance to show as **Available**. Then choose the DB instance name to show its details.
17. 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

18. Complete Install a web server on your EC2 instance.

Install a web server on your EC2 instance

Install a web server on the EC2 instance you created in Launch an EC2 instance. The web server connects to the Amazon RDS DB instance that you created in Create a DB instance.

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.
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
```

3. 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
```

4. Install the Apache web server.

```
sudo yum install -y httpd
```

5. 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). 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.

6. 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
```

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

```
exit
```

3. 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
```

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

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

5. 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 {} \;
```

6. 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 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
```

2. 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
```

3. 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');

?>
```

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

```
cd /var/www/html
```

6. 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
```

7. 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;
}
?>
```

8. Save and close the `SamplePage.php` file.
9. 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.

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. You don't need to create a final snapshot.
- To terminate an Amazon EC2 instance, follow the instruction in Terminate your instance 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.
- 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.

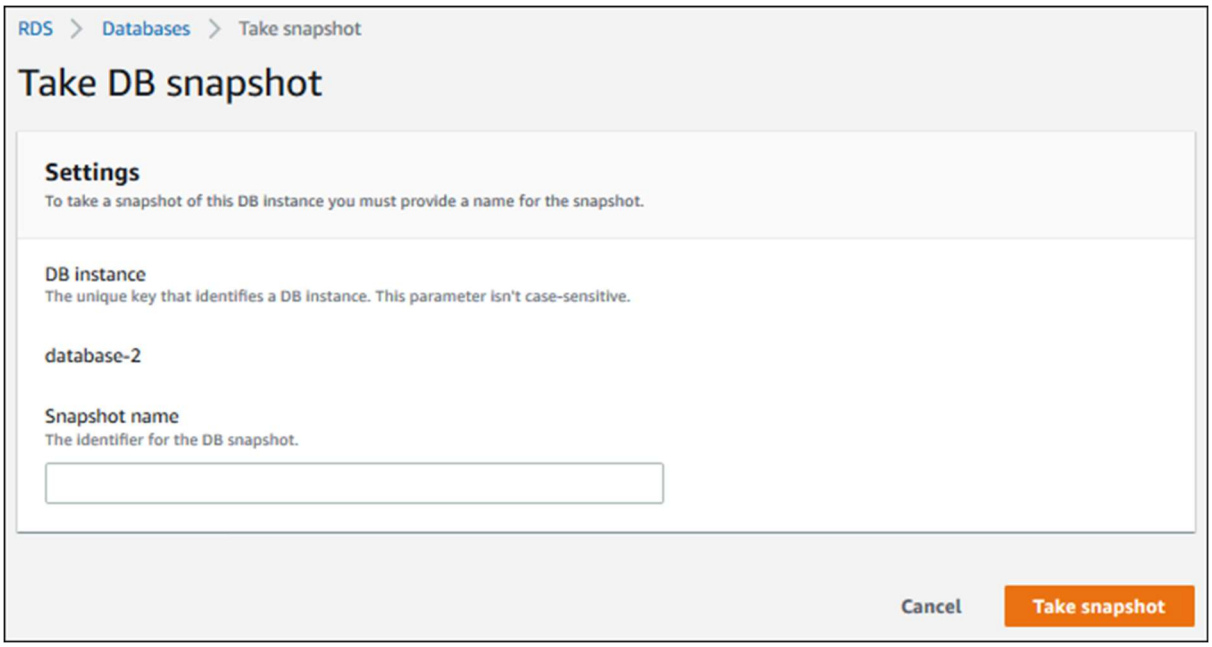
Unlike automated backups, manual snapshots aren't subject to the backup retention period. Snapshots don't expire.

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.

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.



7. 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.

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.
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.

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

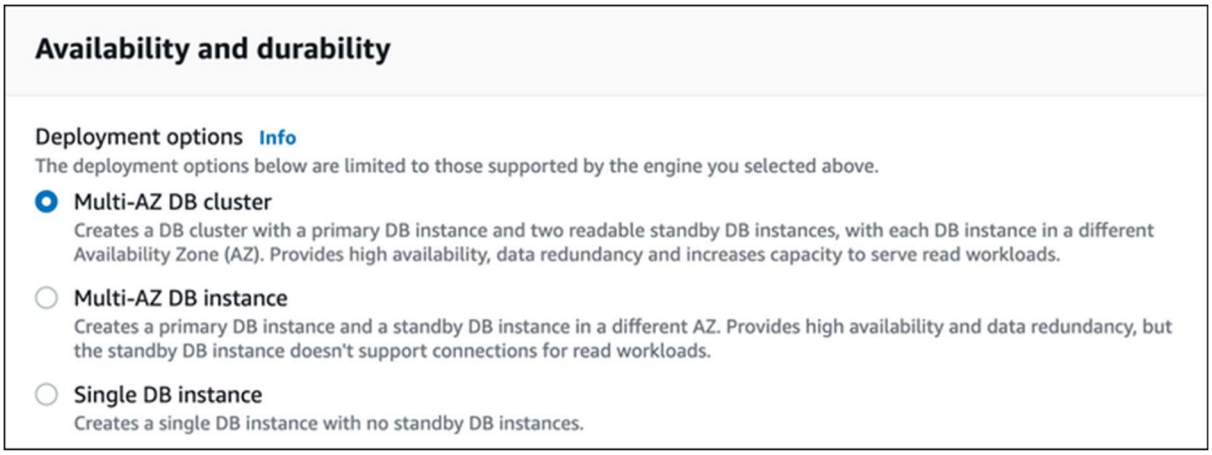
You can restore a snapshot to a Multi-AZ DB cluster using the AWS Management Console, the AWS CLI, or the RDS API. You can restore each of these types of snapshots to a Multi-AZ DB cluster:

- 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**.



6. For **DB cluster identifier**, enter the name for your restored Multi-AZ DB cluster.
7. For the remaining sections, specify your DB cluster settings. For information about each setting, see Settings for creating Multi-AZ DB clusters.
8. 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/

1. Select region EU – Frankfurt (*Mumbai does not support 3 AZs*)
2. Create a VPC
   - 192.168.0.0/16
3. Create 3 Subnets in 3 AZs
   - 192.168.1.0/24
   - 192.168.2.0/24
   - 192.168.3.0/24
4. Create IGW
5. Attach IGW to VPC
6. Create a single Route Table for the 3 public subnets
   i. Create RT for the VPC
   ii. Associate the 3 subnets
   iii. Create route for internet access
      - 0.0.0.0/0 – IGW created earlier
7. Create 3 Windows EC2 instances in each of the 3 AZs

   Server 01:
   i. Windows
   ii. t2.micro
   iii. New key pair
   iv. Select our VPC
   v. Subnet 01
   vi. Auto assign public IP: Enable
   vii. New security group
      - RDP, HTTP

   Server 02:
   i. Windows
   ii. t2.micro
   iii. select key pair create earlier
   iv. Select our VPC
   v. Subnet 02
   vi. Auto assign public IP: Enable
   vii. Select security group created earlier
      - RDP, HTTP

   Server 03:
   i. Windows
   ii. t2.micro
   iii. select key pair created earlier
   iv. Select our VPC
   v. Subnet 03
   vi. Auto assign public IP: Enable

vii. Select security group created earlier
    - RDP, HTTP

8. 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 !!
```

9. 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.

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.

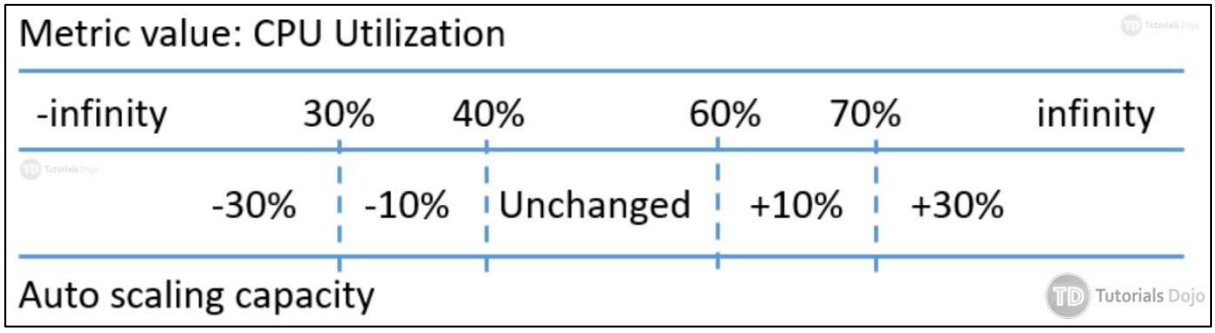
When EC2 Auto Scaling was first introduced, this was the only scaling policy supported. It does not provide any fine-grained control to scaling in and scaling out.

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 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
```

7. Now click on **Create Launch Template.**
8. 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
```

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

```
$ sudo yum install stress -y
```

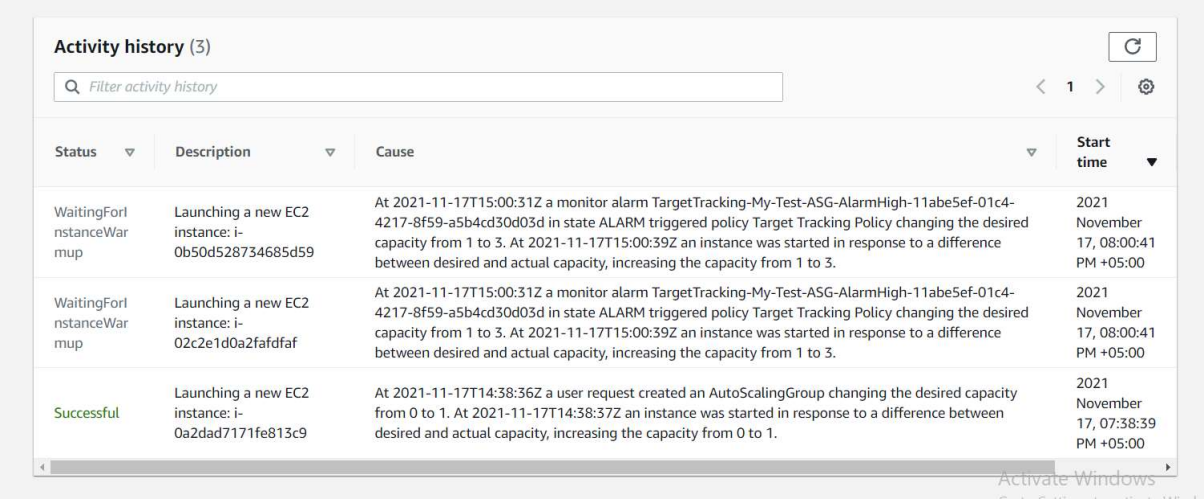
7. After that, we will type in the following command in order to increase CPU utilization.

```
$ stress -c 4
```

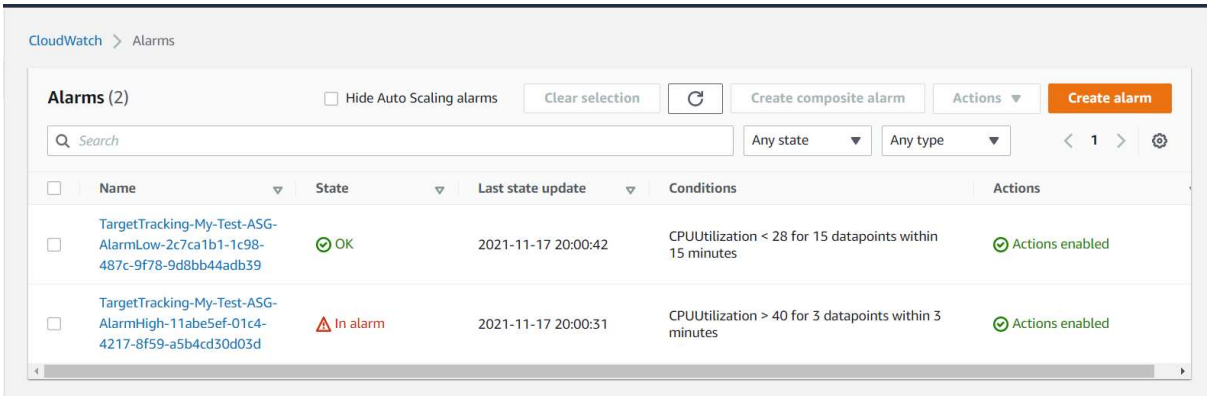
Alternatively:

```
$ while true; do /bin/true; done
```

8. 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.



9. Can also see new instances being created on EC2 instances dashboard.
10. In the CloudWatch monitoring, we can view the CPU utilization skyrocketing.
11. 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.



12. 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.
13. 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**.
14. In the **CloudWatch Alarms**, now we can see that AlarmLow is in the **InAlarm** state.
15. 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) 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.

8. To add another step, choose **Add step** and then define the amount by which to scale and the lower and upper bounds of the step relative to the alarm threshold.
9. Choose **Create**.

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.
- 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.
- 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.
- 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.

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 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.

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://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
- Hard disk drive (HDD) volumes
- Previous generation volumes

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.

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. 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 mentions that Amazon Instance Store is designed for temporary storage for EC2. Things like Buffer, Cache,, 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** 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 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, 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.

```
[ec2-user@                ~]$ sudo lsblk -f
NAME          FSTYPE LABEL UUID                                 MOUNTPOINT
nvme1n1
nvme0n1
├─nvme0n1p1   xfs    /     111667cd-96f8-41ba-bb2d-7c558ae6ad71 /
└─nvme0n1p128
```

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

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

5. 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
```

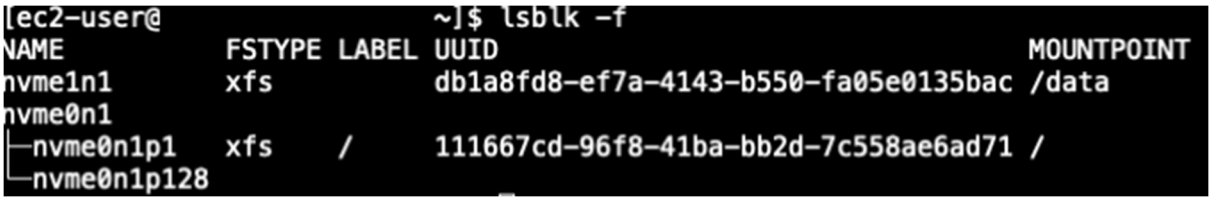
6. To mount the data volume at the directory you just created, use the following command with the data volume name (`/dev/nvme1n1`) and the mount point directory name (`/data`).

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

7. 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.
8. 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.



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.

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.

```
[ec2-user@                ]$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
devtmpfs       devtmpfs  462M     0  462M   0% /dev
tmpfs          tmpfs     471M     0  471M   0% /dev/shm
tmpfs          tmpfs     471M  412K  470M   1% /run
tmpfs          tmpfs     471M     0  471M   0% /sys/fs/cgroup
/dev/nvme0n1p1 xfs       8.0G  1.6G  6.5G  20% /
/dev/nvme1n1   xfs       8.0G   41M  8.0G   1% /data
tmpfs          tmpfs      95M     0   95M   0% /run/user/1000
```

4. 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
```

5. 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.

```
[ec2-user@                ]$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
devtmpfs       devtmpfs  462M     0  462M   0% /dev
tmpfs          tmpfs     471M     0  471M   0% /dev/shm
tmpfs          tmpfs     471M  468K  470M   1% /run
tmpfs          tmpfs     471M     0  471M   0% /sys/fs/cgroup
/dev/nvme0n1p1 xfs       8.0G  1.6G  6.5G  20% /
/dev/nvme1n1   xfs        16G   50M   16G   1% /data
tmpfs          tmpfs      95M     0   95M   0% /run/user/1000
tmpfs          tmpfs      95M     0   95M   0% /run/user/0
```

Route 53

AWS Solution Architect – Class 6 presentation

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/getting-started-s3.html#getting-started-find-domain-name

Configure Health Check for EC2 Instances

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
3. 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 if 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)

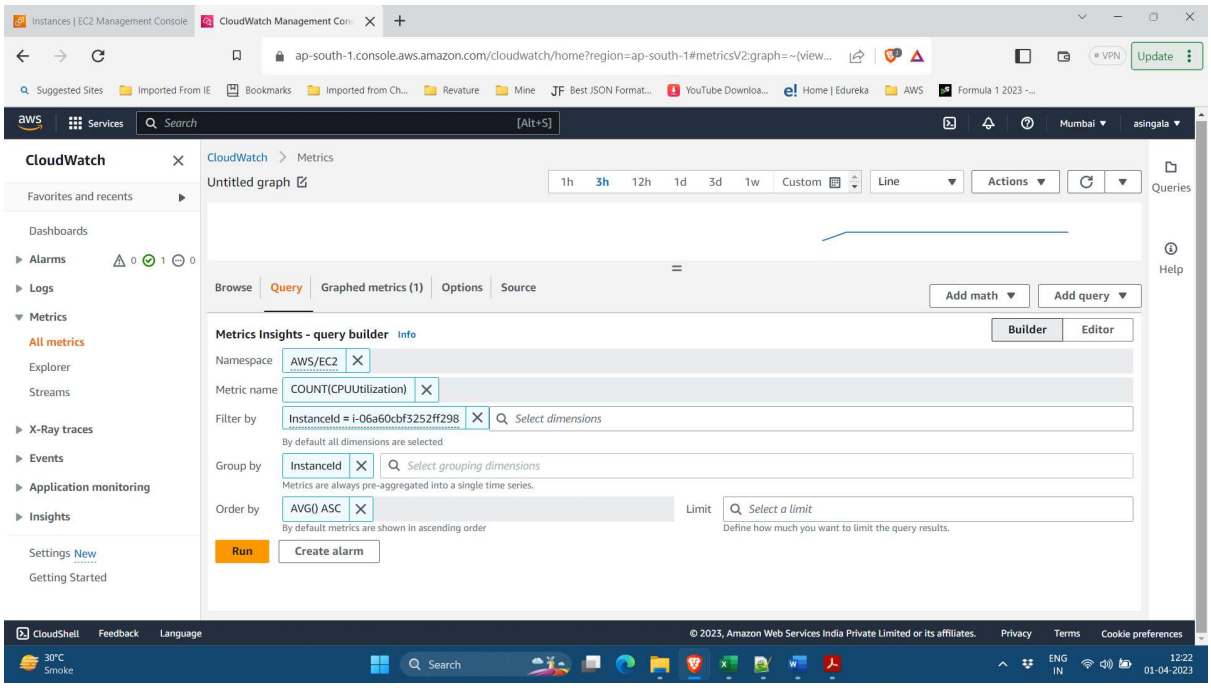
AWS CloudWatch – Monitoring

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
19. Email id
20. Create Topic
21. Select existing topic
22. Next
23. Alarm name
24. Review -> Create Alarm
25. Check if “Actions enabled”
26. Select alarm
27. Select Actions tab
28. Now, run a CloudWatch Metrics Insights query using the CloudWatch console
29. In the navigation pane on the left, select Metrics -> All Metrics -> Browse
30. “View automatic dashboard” under EC2
31. In the navigation pane on the left, select Metrics -> All Metrics -> Query
32. Namespace: AWS/EC2
33. Metric name: COUNT(CPUUtilization)
34. Filter by: InstanceId = <EC2 instance id>
35. Group by: InstanceId
36. Order by: AVG() ASC
37. Run
38. In the navigation pane on the left, select Metrics -> All Metrics -> Graphed Metrics

39. Remove the query created by clicking on the “X” under Actions
40. 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.

5. Select the box for the **EstimatedCharges** metric, and then choose **Select metric**.
6. For **Statistic**, choose **Maximum**.
7. For **Period**, choose **6 hours**.
8. For **Threshold type**, choose **Static**.
9. For **Whenever EstimatedCharges is . . .**, choose **Greater**.
10. 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?.

> **Note:** After you define a threshold value, the preview graph displays your estimated charges for the current month.

11. 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.
12. Choose Next.
13. 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.
14. Choose Next.
15. Under Name and description, enter a name for your alarm.
    - (Optional) Enter a description of your alarm.
16. 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 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.

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,defa
ultValue=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.

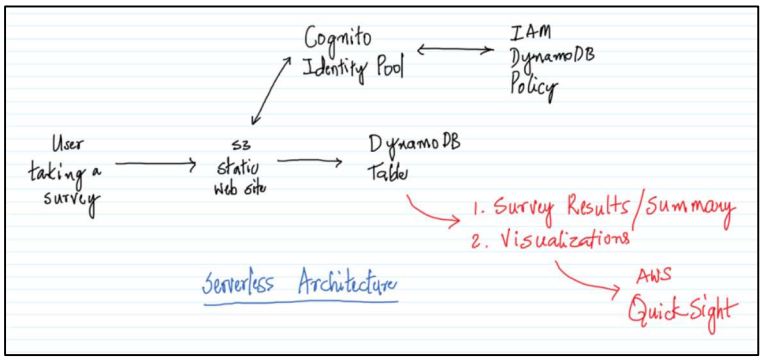
Use case:

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`.

13. In a new browser tab, navigate to IAM -> Roles
14. Filter for the **Cognito** rules and click on the Role which ends with `PoolUnauth_Role`.
15. Select Add permissions -> Attach policies
16. Select the `AmazonDynamoDBFullAccess` Policy and click on `Attach policy` or “Add permissions”.
17. Navigate to **S3 Management Console** and create a **Bucket**.
18. Name: ajs-survey-website-01
19. Uncheck “**Block *all* public access**” -> Acknowledge warning
20. Versioning Disabled
21. Create bucket
22. Select the bucket -> Properties tab
23. Towards the end of the page click on Edit for `Static website hosting`.
24. Select Enable for `Static website hosting`. For the `Index document` enter `survey.html` and for the Error document` enter `error.html`.
25. Save changes
26. 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.
27. In the Permissions Tab, click on Edit for the `Bucket policy`.
28. 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/*"
            ]
        }
    ]
}
```

29. Save changes
30. 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
31. 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**.
32. If everything works fine, then the message `Thanks for helping with the survey` should be displayed.
33. Navigate to the **DynamoDB Management Console** -> **Tables**
34. Click on the **survey** table name -> Explore table items
35. 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.
36. 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 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 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.
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. 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;
```

2. Test it using the following JSON:

```
{
  "sentence": "HELLO WORLD THIS IS AN EXAMPLE WITH LAMBDA"
}
```

3. Navigate to API Gateway -> Create API -> REST API (*not REST API private*) -> Build
4. Protocol: REST
5. Create new API: New API
6. API name: simpleAPI
7. Create API
8. Click on the API -> Resources -> Actions -> Create Resource
9. Resource name: lower-case
10. Uncheck CORS
11. Create Resource
12. Select the resource -> Action -> Create Method -> POST (*from dropdown*) -> Save (*tick mark*)
13. Integration type: Lambda function
14. Lambda Function: toLowerCase
15. Save -> OK
16. Actions -> Deploy API
17. Stage: New Stage
18. Stage name: test
19. Deploy
20. Save changes
21. From left pane -> Stages
22. Expand the stage, select the method
23. Copy the “Invoke URL”
24. Open Postman
25. Select POST and enter the URL
26. Select Body -> Raw -> JSON and add this JSON:

```
{
  "sentence": "HELLO WORLD THIS IS AN EXAMPLE WITH LAMBDA"
}
```

27. Send
28. Should get result as:

```
{
    "statusCode": 200,
    "body": "\"hello world this is an example with lambda\""
}
```

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
```

2. Installed in C:\Program Files\Amazon\AWSCLIV2\
3. 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
```

4. Make sure you have created a non-root user on AWS with **AdministratorAccess** permissions.
5. Configure user with AWS CLI

```
C:\> aws configure
```

6. 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
```

7. 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.
    a. 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

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

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

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

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

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.

This example is for the short-term credentials from AWS Identity and Access Management. For more information, see Authenticating using short-term credentials.

Credentials file

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token =
IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVER
YLONGSTRINGEXAMPLE

[user1]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
aws_session_token =
fcZib3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVER
YLONGSTRINGEXAMPLE
```

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

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. 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).

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.

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.

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. 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.

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.

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.

Step 5: Run the Sample

Type the following command to run the sample.

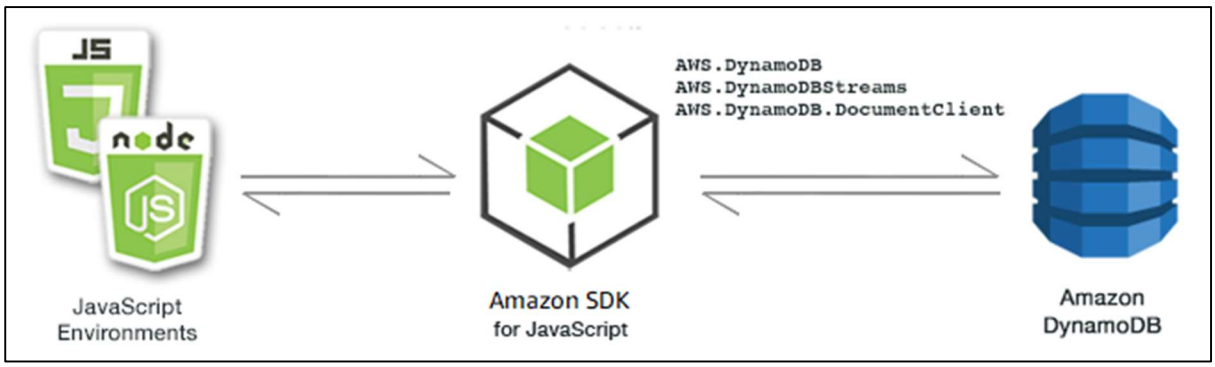
```
node sample.js
```

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.

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.



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`, `Class: AWS.DynamoDBStreams`, and `Class: AWS.DynamoDB.DocumentClient` 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
- listTables
- describeTable
- deleteTable

Prerequisite Tasks

To set up and run this example, first complete these tasks:

- Install Node.js. For more information, see the Node.js website.
- 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.
- 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: 'REGION'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {
  AttributeDefinitions: [
    {
      AttributeName: 'CUSTOMER_ID',
      AttributeType: 'N'
    },
```

```
    {
      AttributeName: 'CUSTOMER_NAME',
      AttributeType: 'S'
    }
  ],
  KeySchema: [
    {
      AttributeName: 'CUSTOMER_ID',
      KeyType: 'HASH'
    },
    {
      AttributeName: 'CUSTOMER_NAME',
      KeyType: 'RANGE'
    }
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1
  },
  TableName: 'CUSTOMER_LIST',
  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.

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: '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.

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: '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.

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: '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.

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
- getItem
- deleteItem

Prerequisite Tasks

To set up and run this example, first complete these tasks:

- Install Node.js. For more information, see the Node.js website.
- 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.
- Create a DynamoDB table whose items you can access. For more information about creating a DynamoDB table, see Creating and Using Tables in DynamoDB.

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: 'REGION'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {
  TableName: 'CUSTOMER_LIST',
  Item: {
    'CUSTOMER_ID' : {N: '001'},
    'CUSTOMER_NAME' : {S: 'Richard Roe'}
  }
};

// Call DynamoDB to add the item to 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.

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: 'REGION'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({apiVersion: '2012-08-10'});

var params = {
  TableName: 'TABLE',
  Key: {
    'KEY_NAME': {N: '001'}
  },
  ProjectionExpression: 'ATTRIBUTE_NAME'
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

To run the example, type the following at the command line.

```
node ddb_getitem.js
```

This sample code can be found here on GitHub.

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: 'TABLE',
  Key: {
    'KEY_NAME': {N: 'VALUE'}
  }
};

// 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.

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
- scan

Prerequisite Tasks

To set up and run this example, first complete these tasks:

- Install Node.js. For more information, see the Node.js website.
- 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.

- Create a DynamoDB table whose items you can access. For more information about creating a DynamoDB table, see Creating and Using Tables in DynamoDB.

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.

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.

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`
- `createTags`

Prerequisite Tasks

To set up and run this example, first complete these tasks.

- Install Node.js. For more information, see the Node.js website.
- 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.
- Create a key pair. For details, see Working with Amazon EC2 Key Pairs. 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.

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.

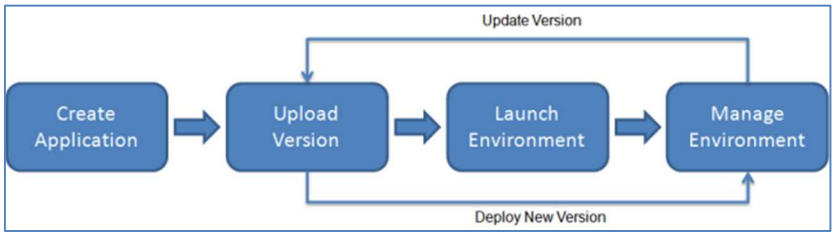
Elastic Beanstalk supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby. When you deploy your application, Elastic Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, to run your application.

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.

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.



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. A backend environment that pulls tasks from an Amazon Simple Queue Service (Amazon SQS) queue runs in a worker environment tier.

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

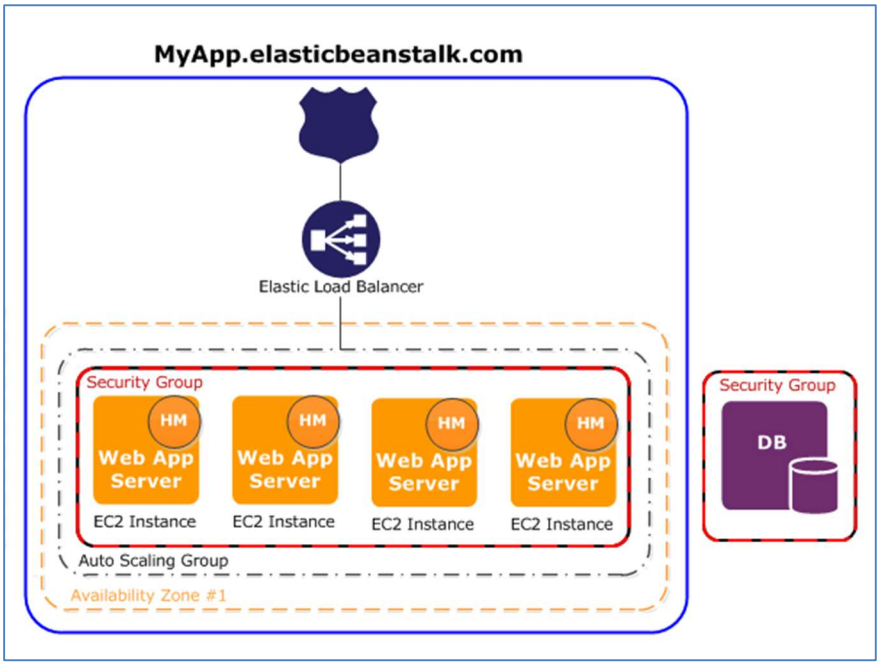
A *saved configuration* is a template that you can use as a starting point for creating unique environment configurations. You can create and modify saved configurations, and apply them to environments, using the Elastic Beanstalk console, EB CLI, AWS CLI, or API. The API and the AWS CLI refer to saved 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.



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 to an Elastic Load Balancing URL—something like `abcdef-123456.us-west-2.elb.amazonaws.com`—by using a CNAME record. Amazon Route 53 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. 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

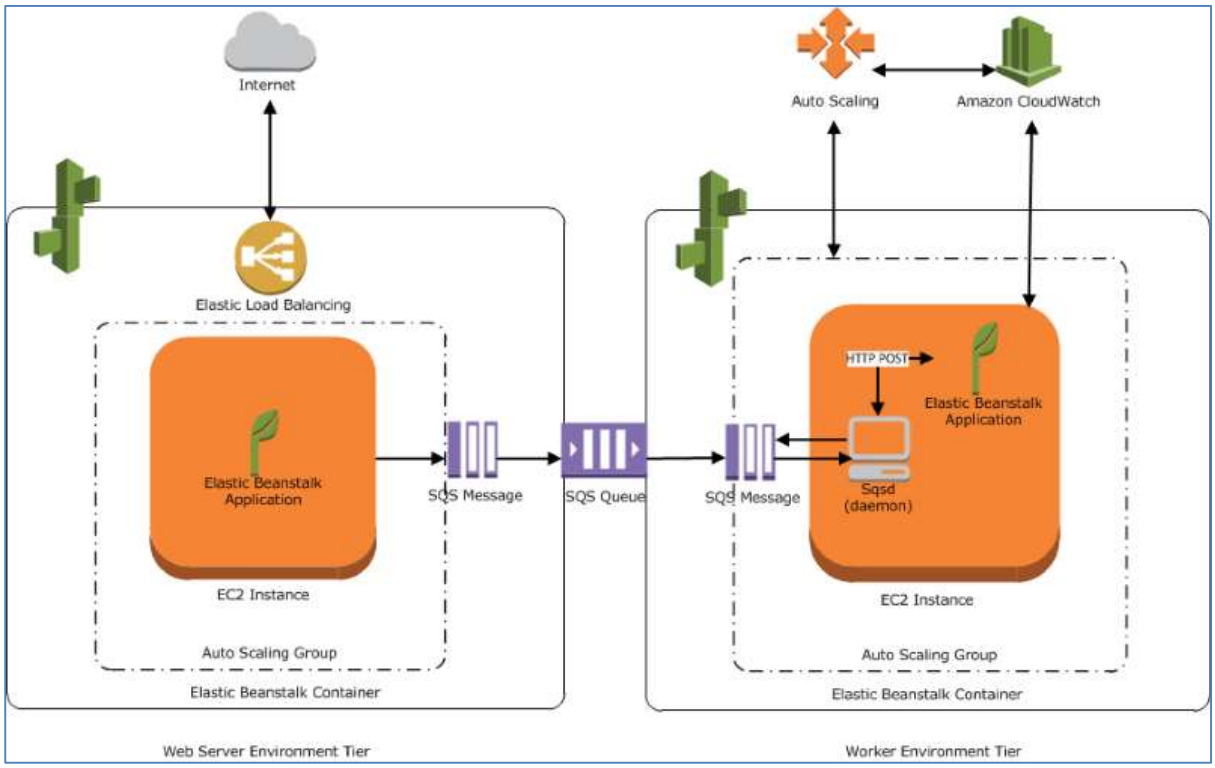
The host manager reports metrics, errors and events, and server instance status, which are available via the Elastic Beanstalk console, APIs, and CLIs.

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.



Amazon CloudWatch is used for alarms and health monitoring. For more information, go to Basic health reporting.

For details about how the worker environment tier works, see Elastic Beanstalk worker environments.

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

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, API, or AWS CLI.

For more information about creating, updating, or deleting stacks, see Working with stacks.

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.

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*).
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

```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 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 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 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 function is used to get the ID of the security group with the logical name `ELBSecurityGroup`.

AWS::ElasticLoadBalancingV2::TargetGroup 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 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 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 function is used to get the ID of the security group with the logical name `ELBSecurityGroup`.

AWS::EC2::LaunchTemplate 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 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 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.
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 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.

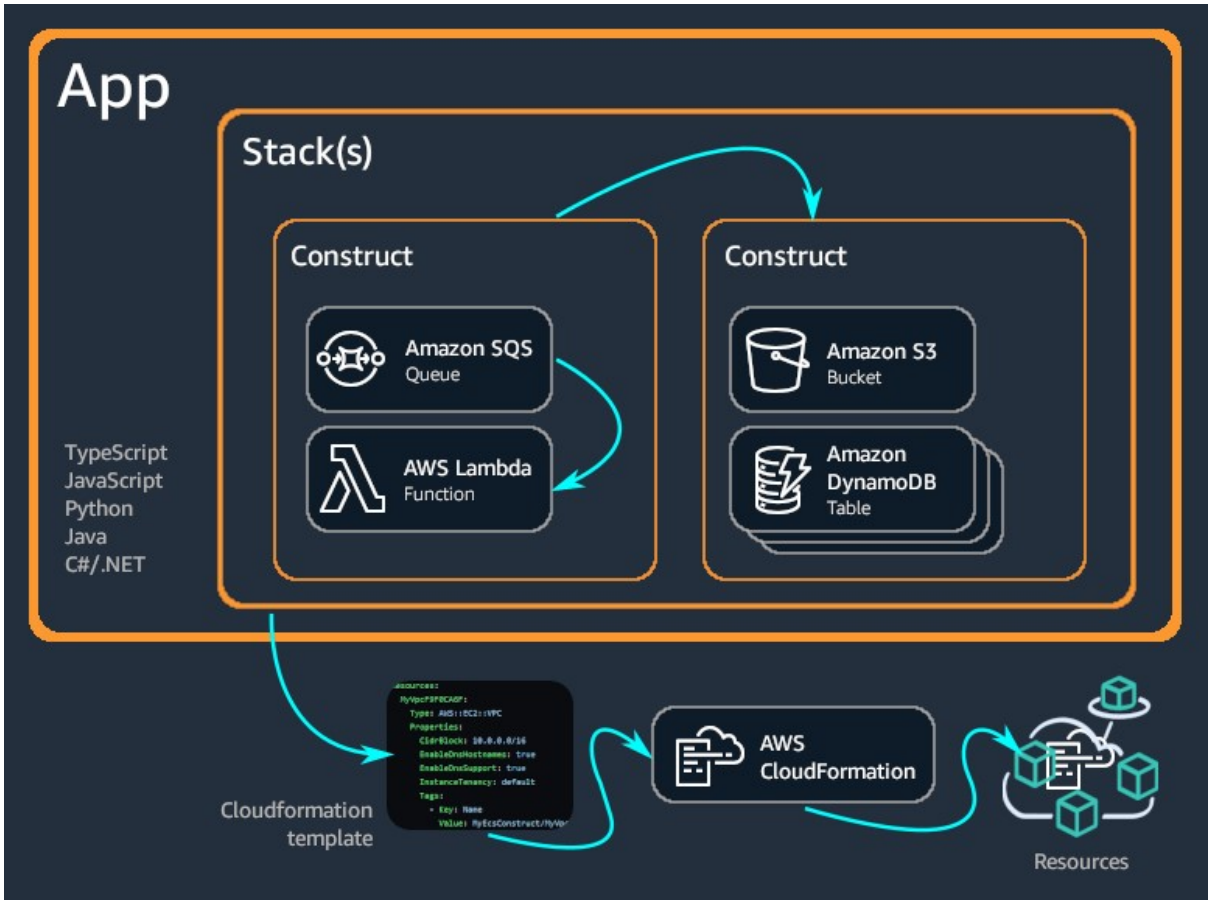
Connect your AWS resources together (even across stacks) and grant permissions using simple, intent-oriented APIs.

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. You compose these together into Stacks and Apps.



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).

What is AWS Fargate?

AWS Fargate is a technology that you can use with Amazon ECS to run 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
        {
```

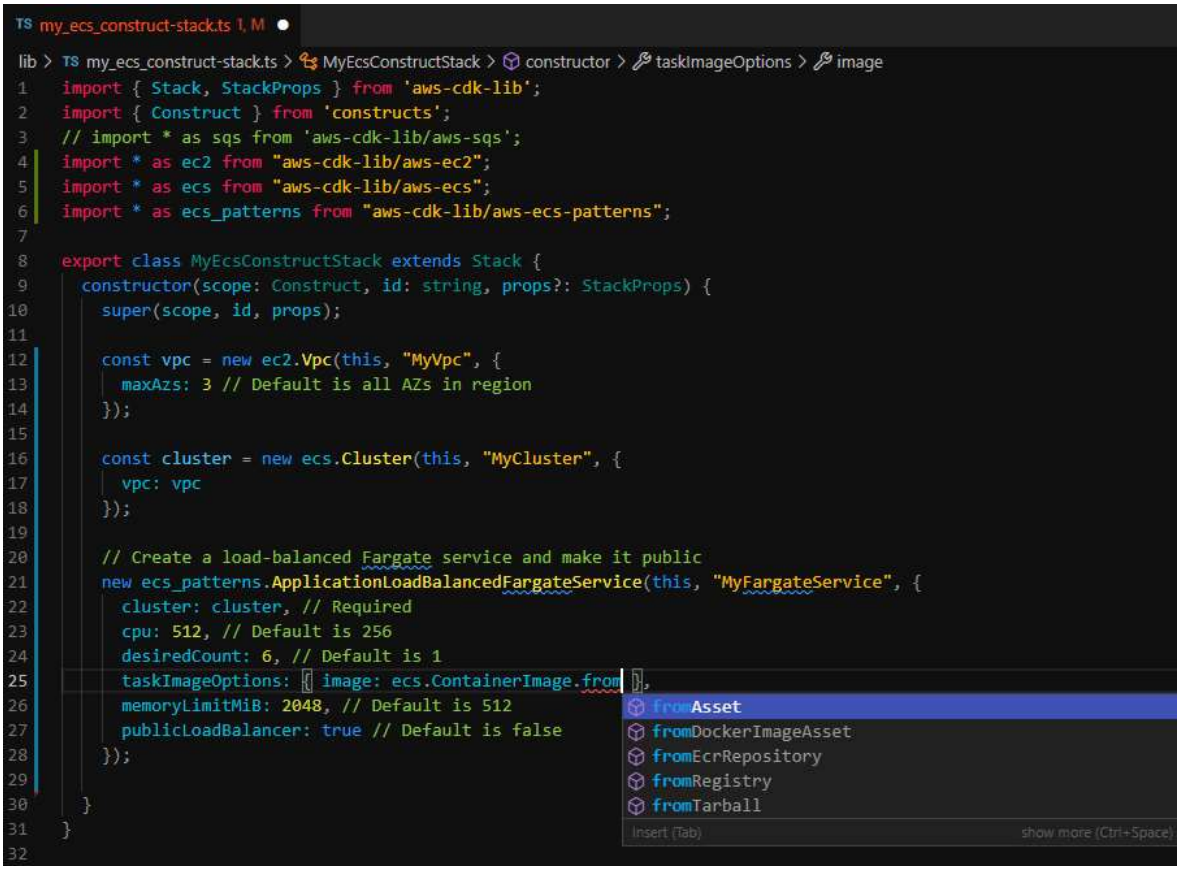
```
            Vpc = vpc
        });

        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; deploying the AWS CDK app produces more than 50 resources of the following types.

- AWS::EC2::EIP
- AWS::EC2::InternetGateway
- AWS::EC2::NatGateway
- AWS::EC2::Route
- AWS::EC2::RouteTable
- AWS::EC2::SecurityGroup
- AWS::EC2::Subnet
- AWS::EC2::SubnetRouteTableAssociation
- AWS::EC2::VPCGatewayAttachment
- AWS::EC2::VPC
- AWS::ECS::Cluster
- AWS::ECS::Service
- AWS::ECS::TaskDefinition
- AWS::ElasticLoadBalancingV2::Listener
- AWS::ElasticLoadBalancingV2::LoadBalancer
- AWS::ElasticLoadBalancingV2::TargetGroup
- AWS::IAM::Policy
- AWS::IAM::Role
- AWS::Logs::LogGroup

And let's not forget... code completion within your IDE or editor!


my_ecs_construct-stack.ts 1, M
lib > my_ecs_construct-stack.ts > MyEcsConstructStack > constructor > taskImageOptions > image
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
// import * as sqs from 'aws-cdk-lib/aws-sqs';
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as ecs from "aws-cdk-lib/aws-ecs";
import * as ecs_patterns from "aws-cdk-lib/aws-ecs-patterns";
export class MyEcsConstructStack extends Stack {
constructor(scope: Construct, 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.from },
memoryLimitMiB: 2048, // Default is 512
publicLoadBalancer: true // Default is false
});
}
}
fromAsset
fromDockerImageAsset
fromEcrRepository
fromRegistry
fromTarball
Insert (Tab)
show more (Ctrl+Space)

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**. When git is used in combination with a continuous integration / continuous deployment (CI/CD) tool such as **GitHub Actions** or **Jenkins**, 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**. 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** 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.