

# Access Amazon S3 from a VPC

By Haroon Zaman | December 2025

## Introduction

In this project, I learned how to allow resources inside a **Virtual Private Cloud (VPC)** to securely interact with **Amazon S3**. By default, resources in a VPC cannot access AWS services unless proper permissions and network configurations are in place. This project focuses on enabling that access in a controlled and secure way.

The goal of this project was to understand how EC2 instances communicate with AWS services like S3, how access permissions are granted using AWS credentials, and how traffic flows from a VPC to external AWS services.

In this project, I performed the following tasks:

- Set up a **VPC with an EC2 instance**
- Used **AWS access keys** to give the EC2 instance permission to interact with AWS services
- Accessed and interacted with **Amazon S3 directly from the EC2 instance**
- Observed how network and security configurations allow private resources to reach AWS-managed services

This project helped me understand how AWS services work together, how identity and access management controls permissions, and how cloud resources securely communicate with services outside the VPC while staying within AWS's infrastructure.

The screenshot shows two main sections of the AWS VPC configuration interface.

**VPC Details Page:** This section displays various configuration details for the VPC. Key settings include:

- VPC ID:** vpc-0a25759383a846984
- State:** Available
- Tenancy:** default
- Default VPC:** No
- Network Address Usage metrics:** Disabled
- Encryption control mode:** -
- Block Public Access:** Off
- DHCP option set:** dopt-0dc748262ab127598
- IPv4 CIDR:** 10.0.0.0/16
- Route 53 Resolver DNS Firewall rule groups:** -
- DNS hostnames:** Enabled
- Main route table:** rtb-0267ba2b34258101c
- IPv6 pool:** -
- Owner ID:** 577721963177

**Resource Map:** This section provides a visual overview of the VPC structure, showing its components and their relationships:

- VPC:** Your AWS virtual network (VPC\_1-vpc)
- Subnets (1):** Subnets within this VPC (eu-north-1a, VPC\_1-subnet-public1-eu-north-1a)
- Route tables (2):** Route network traffic to resources (VPC\_1-rtb-public, rtb-0267ba2b34258101c)
- Network Connections (1):** Connections to other networks (VPC\_1-igw)

At the bottom of the Resource Map section, there is a link to "Show all details".

# Create VPC and EC2 Instance

- I created a new VPC and named it **VPC\_1**.
- I assigned the IPv4 CIDR block **10.0.0.0/16** to provide a large private IP range.
- I selected **one Availability Zone** for a simple setup.
- I created **one public subnet** and **no private subnet**, because this project only requires a public EC2 instance to access Amazon S3.
- I did not configure a **NAT Gateway**, since NAT is only needed when private subnets require outbound internet access.
- After reviewing the settings, I created the VPC.

The screenshot shows the AWS CloudWatch Metrics console with a search bar at the top. Below the search bar, there are two main sections: 'Metrics' and 'Logs'. Under 'Metrics', there is a table with columns for 'Metric Name', 'Unit', 'Value', and 'Timestamp'. The table contains several rows of data, including metrics like 'AWS Lambda Function Invocations', 'AWS Lambda Function Errors', and various CPU and Memory metrics. The 'Logs' section below has a search bar and a table with columns for 'Log Stream', 'Timestamp', and 'Message'. It displays log entries from an EC2 instance, including logs related to AWS Lambda function invocations and memory usage.

- Next, I created an EC2 instance and named it **Instance\_1**.
- I selected **Amazon Linux** as the operating system.
- I chose the **t3 instance type**, which is suitable for lightweight testing and learning.
- I did **not create a key pair**, as SSH access was not required for this task.
- In the network settings, I selected **VPC\_1** and its public subnet.
- I enabled **Auto-assign Public IP** so the instance could reach AWS services.
- I created a new **Security Group** and allowed **SSH**.
- Finally, I launched the EC2 instance successfully.

The screenshot shows the AWS CloudWatch Metrics Insights query results table. The table has columns for 'Time' and 'Metric'. The data shows a single metric named 'AWS Lambda Function Invocations' with a value of 1. The time range is from 2023-09-11T00:00:00Z to 2023-09-11T00:01:00Z. The table also includes a 'Metrics' column with a link to the original metric definition.

Time	Metric	Metrics
2023-09-11T00:00:00Z - 2023-09-11T00:01:00Z	AWS Lambda Function Invocations: 1	<a href="#">AWS Lambda Function Invocations</a>

# AWS Access Keys and Credentials

- After creating the instance, I connected to it and ran the command:  
`aws s3 ls`
  - This returned an error saying “**Unable to locate credentials**”, which means the instance did not yet have permission to access AWS services.
  - To fix this, I ran `aws configure`, which requires an **AWS Access Key ID**.
- 

## What is an Access Key ID?

- An **Access Key ID** is part of an AWS credential.
  - AWS credentials are similar to a **username and password**.
  - The **Access Key ID** acts like the **username**, and the **Secret Access Key** acts like the password.
  - You don’t get an access key automatically — it must be created through **AWS IAM**.
- 

## Difference Between Access Keys and Key Pairs

- **Key Pairs** are used for **logging into EC2 instances via SSH**.
- **Access Keys** are used by **applications and services** (like AWS CLI) to authenticate and interact with AWS resources such as S3.
- In simple terms:
  - **Key Pair → Login to EC2**
  - **Access Key → Access AWS services**

The screenshot shows the AWS IAM User Details page for a user named "Haroon".

**Security Credentials:**

- Console access:** Enabled without MFA. Last console sign-in was today.
- Access key 1:** ARN: arn:aws:siam:377721963177:user/Haroon. Status: Active. Created today. No usage.
- Access key 2:** Create access key.

**Multifactor authentication (MFA):** 0 devices assigned. Assign MFA device.

**Access keys (1):** ARN: ARKAVP4P3YKUZRKD7QNV. Status: Active. Description: Default access key. Actions: Create access key, Delete, Resync, Assign MFA device.

# Creating an IAM User for Access Keys

- To create an **Access Key**, I first needed an **IAM user**, because access keys are managed at the user level.
- Since I did not have any IAM user already created, I created one myself.
- I created a new IAM user and named it **Haroon**.
- I granted this user **Administrator permissions** so it would have full access to AWS services for this project.
- After creating the user, I was able to manage and generate **Access Keys** for it.

The screenshot shows the AWS IAM User Haroon console page. At the top, there's a summary section with ARN (arn:aws:iam:377721963177:user/Haroon), creation date (December 15, 2025, 14:37 (UTC+03:00)), and a single access key listed. Below this is a navigation bar with tabs: Permissions, Groups, Tags, Security credentials (which is selected), and Last Accessed. Under the Security credentials tab, there are sections for Console sign-in (with a link to https://377721963177.signin.aws.amazon.com/console) and Multi-factor authentication (MFA) (0). The MFA section has buttons for Remove, Resync, and Assign MFA device. The final section, 'Access keys (0)', is highlighted with a red border. It contains instructions about using access keys for programmatic calls, a note about best practices, and a 'Create access key' button. There are also 'Create access key' buttons in the other sections.

- After creating the IAM user, I opened the user **Haroon** in the IAM console.
- I scrolled down to the **Access keys** section.
- From there, I clicked on **Create access key** to generate new credentials for AWS CLI access.
- After clicking **Create access key**, a new window opened.

In Step 1, I had to choose the **use case** for the access key.

- The available use case options were:
- **Command Line Interface (CLI)**  
Used when you want to access AWS services using the AWS CLI commands.
- **Local code**  
Used when application code running on your local machine needs to access AWS resources.
- **Application running on an AWS compute service**  
Used when applications running on EC2, ECS, or Lambda need AWS access.
- **Third-party service**  
Used when an external tool or service needs permission to manage or monitor AWS resources.

> Create access key

Step 1  
 Access key best practices & alternatives  
 Step 2 - optional  
 Set description tag  
 Step 3  
 Retrieve access keys

**Access key best practices & alternatives** Info  
 Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

**Use case**

- Command Line Interface (CLI)  
 You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code  
 You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service  
 You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- Third-party service  
 You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS  
 You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
- Other  
 Your use case is not listed here.

**Alternatives recommended**

- Use AWS CLI V2 and the `aws login` command to use your existing console credentials in the CLI. [Learn more](#)
- Use AWS CloudShell, a browser-based CLI, to run commands. [Learn more](#)

**Confirmation**

I understand the above recommendation and want to proceed to create an access key.

[Cancel](#) [Next](#)

- **Application running outside AWS**  
 Used when workloads running outside AWS (on-prem or another cloud) need AWS access.
- **Other**  
 Used if your use case does not fit any of the listed options.
- I selected **Command Line Interface (CLI)** because I planned to use the AWS CLI from my EC2 instance.
- I checked the **confirmation box** and clicked **Next**.

Step 1  
 Access key best practices & alternatives  
 Set description tag  
 Step 2 - optional  
 Retrieve access keys

**Set description tag - optional** Info  
 The description for this access key will be attached to this user as a tag and shown alongside the access key.

**Description tag value**  
 Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.  
  
Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: \_ . : / = + - @

[Cancel](#) [Previous](#) [Create access key](#)

- In the next step, I added a **description** to help identify the access key later.
- After setting the description, I clicked **Create access key** to generate the credentials.
- After creating the access key, AWS generated an **Access Key ID** and a **Secret Access Key**.
- These credentials are used together to authenticate with AWS services.
- I was given the option to **download the keys as a .csv file** for safe storage.
- This is the only time the **Secret Access Key** is visible, so it must be saved securely.

Step 1  
 Access key best practices & alternatives

Step 2 - optional  
 Set description tag

Step 3  
 Retrieve access keys

### Retrieve access keys Info

**Access key**  
 If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
	***** Show

**Access key best practices**

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#) [Done](#)

```

Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Mon Dec 15 11:21:38 2025 from 13.40.4.203
[ec2-user@ip-10-0-12-239 ~]$ aws s3 ls
[ec2-user@ip-10-0-12-239 ~]$ aws configure
[ec2-user@ip-10-0-12-239 ~]$ aws config
[ec2-user@ip-10-0-12-239 ~]$ aws s3
AWS Access Key ID [None]: AKIAVH4F3YKUJ3URD7QNV
AWS Secret Access Key [None]: fJfMLnfzq4gvspN0sf32hIHr68c0uM9XzY
Default region name [None]: eu-north-1
Default output format [None]: [ec2-user@ip-10-0-12-239 ~]$ 

```

- After that, I went back to the EC2 instance I had connected to earlier.
- I entered the **Access Key ID** and then the **Secret Access Key** when prompted.
- After providing the credentials, the login and configuration were **successful**.

**Create bucket Info**  
 Buckets are containers for data stored in S3.

#### General configuration

**AWS Region**  
 Europe (Stockholm) eu-north-1

**Bucket type Info**

General purpose  
 Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory  
 Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

**Bucket name Info**  
 storgae-vpc-1

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn more](#)

**Copy settings from existing bucket - optional**  
 Only the bucket settings in the following configuration are copied.  
[Choose bucket](#)

Format: s3://bucket/prefix

#### Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

**Object Ownership**

ACLs disabled (recommended)  
 All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

ACLs enabled  
 Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

**Object Ownership**  
 Bucket owner enforced

## Creating the S3 Bucket

- Next, I created an **Amazon S3 bucket**.
- I searched for **S3** in the AWS console and clicked **Create bucket**.
- I selected the **General purpose** bucket type, which is used for standard object storage like files, backups, and logs.
- I gave the bucket the name **storage\_vpc\_1**.
- For **Object Ownership**, I kept **ACLs disabled**.

- This is recommended by AWS because access should be managed using **IAM policies** instead of ACLs.
- Disabling ACLs improves security and simplifies permission management.

**Block Public Access settings for this bucket**

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

**Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- Block public access to buckets and objects granted through new access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through any access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through new public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- Block public and cross-account access to buckets and objects through any public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

**Bucket Versioning**

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

**Bucket Versioning**

Disable  
 Enable

**Tags - optional**

You can use bucket tags to analyze, manage and specify permissions for a bucket. [Learn more](#)

You can use s3>ListTagsForResource, s3:TagResource, and s3:UntagResource APIs to manage tags on S3 general purpose buckets for access control in addition to cost allocation and resource organization. To ensure a seamless transition, please provide permissions to s3>ListTagsForResource, s3:TagResource, and s3:UntagResource actions. [Learn more](#)

No tags associated with this bucket.

[Add new tag](#)

You can add up to 50 tags.

**Default encryption** [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

**Encryption type** [Info](#)

Secure your objects with two separate layers of encryption. For details on pricing, see DSSE-KMS pricing on the Storage tab of the [Amazon S3 pricing page](#).

Server-side encryption with Amazon S3 managed keys (SSE-S3)  
 Server-side encryption with AWS Key Management Service keys (SSE-KMS)  
 Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

**Bucket Key**

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

Disable  
 Enable

**▼ Advanced settings**

Object Lock

- I left all other settings on **default**.
- Finally, I clicked **Create bucket**, and the S3 bucket was created successfully.

**storgae-vpc-1** [Info](#)

[Objects](#) [Metadata](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

**Objects (0)**

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Find objects by prefix](#)  [Show versions](#)

Name	Type	Last modified	Size	Storage class
No objects You don't have any objects in this bucket.				

[Upload](#)

- After creating the S3 bucket, it was time to upload files.
- I opened the bucket **storage\_vpc\_1**.
- I clicked **Upload** and it opened a new window.

**Upload** [Info](#)

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDKs or Amazon S3 REST API. [Learn more](#)

Drag and drop files and folders you want to upload here, or choose Add files or Add folder.

Files and folders (4 total, 247.2 KB)		Remove	Add files	Add folder
All files and folders in this table will be uploaded.				
<input type="text"/> Find by name				
Name	Folder	Type	Size	
Attack Categories.docx	-	application/vnd.openxmlformats-officedocument.wor...	13.6 KB	
ITIL.docx	-	application/vnd.openxmlformats-officedocument.wor...	21.5 KB	
questions.docx	-	application/vnd.openxmlformats-officedocument.wor...	26.7 KB	
Salary Slip.pdf	-	application/pdf	185.4 KB	

**Destination** [Info](#)

Destination  
[s3://storgae-vpc-1](#)

▶ Destination details  
Bucket settings that impact new objects stored in the specified destination.

▶ Permissions  
Grant public access and access to other AWS accounts.

- In the upload window, I clicked **Add files** and selected some random files from my PC.
- I also had the option to **create a folder** inside the bucket.
- I could choose the **destination location** for the files and **set permissions** if needed.

▼ Properties  
Specify storage class, encryption settings, tags, and more.

**Storage class** [Info](#)

Amazon S3 offers a range of storage classes designed for different use cases. [Learn more](#) or see [Amazon S3 pricing](#)

Storage class	Designed for	Bucket type	Availability Zones	Min storage duration	Min billable object size	Monitoring and auto-tiering fees	Retrieval fees
<input checked="" type="radio"/> Standard	Frequently accessed data (more than once a month) with milliseconds access	General purpose	≥ 3	-	-	-	-
<input type="radio"/> Intelligent-Tiering	Data with changing or unknown access patterns	General purpose	≥ 3	-	-	Per-object fees apply for objects >= 128 KB	-
<input type="radio"/> Standard-IA	Infrequently accessed data (once a month) with milliseconds access	General purpose	≥ 3	30 days	128 KB	-	Per-GB fees apply
<input type="radio"/> One Zone-IA	Recreatable, infrequently accessed data (once a month) with milliseconds access	General purpose or directory	1	30 days	128 KB	-	Per-GB fees apply
<input type="radio"/> Glacier Instant Retrieval	Long-lived archive data accessed once a quarter with instant retrieval in milliseconds	General purpose	≥ 3	90 days	128 KB	-	Per-GB fees apply
<input type="radio"/> Glacier Flexible Retrieval (formerly Glacier)	Long-lived archive data accessed once a year with retrieval of minutes to hours	General purpose	≥ 3	90 days	-	-	Per-GB fees apply
<input type="radio"/> Glacier Deep Archive	Long-lived archive data accessed less than once a year with retrieval of hours	General purpose	≥ 3	180 days	-	-	Per-GB fees apply
<input type="radio"/> Reduced redundancy	Noncritical, frequently accessed data with milliseconds access (not recommended as S3 Standard is more cost effective)	General purpose	≥ 3	-	-	-	Per-GB fees apply

- After scrolling down, I reached the **Storage class** section.
- I selected **Standard**, which is used for frequently accessed data.
- Other storage class options available were:
  - **Intelligent-Tiering**
  - **Standard-IA (Infrequent Access)**
  - **One Zone-IA**
  - **Glacier Instant Retrieval**
  - **Glacier Flexible Retrieval**
  - **Glacier Deep Archive**
- After selecting the storage class, I clicked **Upload**.
- The files were uploaded **successfully** to the S3 bucket.

```
Last login: Mon Dec 15 11:39:52 2025 from 13.48.4.202
[ec2-user@ip-10-0-12-239 ~]$ aws s3 ls
2025-12-15 14:59:50 storgae-vpc-1
[ec2-user@ip-10-0-12-239 ~]$ aws s3 ls s3://storgae-vpc-1
2025-12-15 16:22:01      13928 Attack Categories.docx
2025-12-15 16:22:02      22038 ITIL.docx
2025-12-15 16:22:02      189801 Salary Slip.pdf
2025-12-15 16:22:02      27368 questions.docx
[ec2-user@ip-10-0-12-239 ~]$ █
```

- From inside the EC2 instance in the VPC, after configuring the AWS credentials, I accessed the S3 bucket.
- I ran the command:  
`aws s3 ls s3://storage_vpc_1`
- This command listed **all files and folders** stored inside the S3 bucket.
- The successful output confirmed that the EC2 instance inside the VPC could **access Amazon S3** correctly.

```
[ec2-user@ip-10-0-12-239 ~]$ sudo touch /tmp/test.txt
[ec2-user@ip-10-0-12-239 ~]$ aws s3 ls s3://storgae-vpc-1
2025-12-15 16:22:01      13928 Attack Categories.docx
2025-12-15 16:22:02      22038 ITIL.docx
2025-12-15 16:22:02      189801 Salary Slip.pdf
2025-12-15 16:34:56      0 hello.txt
2025-12-15 16:22:02      27368 questions.docx
[ec2-user@ip-10-0-12-239 ~]$ aws s3 cp /tmp/test.txt s3://storgae-vpc-1
upload: ../../tmp/test.txt to s3://storgae-vpc-1/test.txt
[ec2-user@ip-10-0-12-239 ~]$ aws s3 ls s3://storgae-vpc-1
2025-12-15 16:22:01      13928 Attack Categories.docx
2025-12-15 16:22:02      22038 ITIL.docx
2025-12-15 16:22:02      189801 Salary Slip.pdf
2025-12-15 16:34:56      0 hello.txt
2025-12-15 16:22:02      27368 questions.docx
2025-12-15 16:37:22      0 test.txt
[ec2-user@ip-10-0-12-239 ~]$ █
```

- I also created a file directly from the EC2 instance using the CLI.
- I ran the command:  
`sudo touch /tmp/test.txt`
- This command created a file named **test.txt** inside the `/tmp` directory.
- After creating **test.txt**, I copied it to the S3 bucket.
- I ran the command:  
`aws s3 cp /tmp/test.txt s3://storage_vpc_1`
- After copying the file, I checked the bucket contents again using:  
`aws s3 ls s3://storage_vpc_1`
- The **test.txt** file was listed in the output, confirming that the file was successfully uploaded from the EC2 instance to the S3 bucket.

## Conclusion

In this project, I successfully configured a VPC and EC2 instance to securely access **Amazon S3**. I learned how AWS credentials work, how access keys differ from key pairs, and how IAM users control permissions for AWS services.

By creating an S3 bucket, uploading files through the AWS Console, and interacting with the bucket directly from an EC2 instance using the AWS CLI, I verified that resources inside a VPC can communicate with AWS services when proper identity and access configurations are in place.

This project strengthened my understanding of **IAM, S3, and AWS networking**, and showed how permissions and credentials enable secure access to cloud services without exposing unnecessary public access.