PROJECT REPORT

AWS 3-Tier Web Application

EC2 • S3 • RDS • VPC • Flask • MySQL • Linux • GitHub

A Complete Hands-On Cloud Engineering Project

Field	Details	Field	Details
Name	Kalaiarasi R	Project No.	Project 1 of 5
Email	kalaiarasirajan73@gmail.com	Date	February 2026
GitHub	github.com/Kalaiarasi73	AWS Region	ap-south-1 (Mumbai)
Certification	AWS CCP 2026	Status	■ Completed

Table of Contents

Chapter 1 — Introduction: What is Cloud Computing?

Cloud computing is the delivery of computing services — including servers, storage, databases, networking, and software — over the internet. Instead of owning and maintaining physical computers and servers, companies rent these resources from cloud providers like Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform.

Before cloud computing, if a company wanted to run a website, they had to buy physical servers, set them up in a data center, hire people to maintain them, and pay for electricity and cooling. This was expensive, slow, and difficult to scale. Cloud computing changed all of this by allowing companies to rent exactly the computing power they need, pay only for what they use, and scale up or down instantly.

Amazon Web Services (AWS) is the world's most popular cloud platform. It was launched in 2006 and today provides over 200 services to millions of customers worldwide. Companies like Netflix, Airbnb, NASA, and the Indian government use AWS to power their applications.

What We Built in This Project

In this project, we built a complete 3-tier web application on AWS. A 3-tier architecture separates an application into three layers — the frontend (what users see), the backend (the logic that processes requests), and the database (where data is stored). This separation makes applications more secure, scalable, and maintainable.

Tier	Service Used	Purpose	Technology
Tier 1 — Frontend	Amazon S3	Host the HTML website	HTML, CSS
Tier 2 — Backend	Amazon EC2	Run the application server	Python, Flask
Tier 3 — Database	Amazon RDS	Store application data	MySQL
Networking	Amazon VPC	Private isolated network	Subnets, IGW, Routes

Chapter 2 — Linux Basics: Commands Used in This Project

Linux is a free and open-source operating system that powers most of the world's servers. When we launch an EC2 instance on AWS, it runs Amazon Linux 2023, which is based on Linux. Understanding basic Linux commands is essential for any cloud engineer because almost all server administration is done through the terminal — there is no graphical interface like Windows.

Unlike Windows where you click buttons and icons, in Linux you type commands in a terminal (also called command line or shell). The terminal is extremely powerful — with a single command you can install software, create files, configure a web server, or restart a service. This is why all professional server management is done through the terminal.

Linux Terminal Basics

When you connect to EC2 via SSH, you see this prompt:

```
[ec2-user@ip-10-0-1-170 ~]$
```

This prompt tells you important information. 'ec2-user' is your username, 'ip-10-0-1-170' is the server's hostname, '~' means you are in your home directory, and '$' means you are a regular user (not root/admin). When you become root user, $ changes to #.

Understanding sudo

'sudo' stands for 'Super User Do'. In Linux, not all commands can be run by regular users — some require administrator (root) permissions. When you prefix any command with sudo, it runs that command with admin privileges. This is similar to 'Run as Administrator' in Windows. In this project, we used sudo for installing software and running the Flask app on port 80, because port 80 requires root permission.

```
sudo yum update -y # Update system as admin

sudo yum install python3 -y # Install Python as admin

sudo python3 app.py # Run app with admin privileges
```

Linux Commands Used in This Project

Command	What It Does	Example Used In Project
ssh	Connect securely to remote server	ssh -i key.pem ec2-user@13.232.140.70
sudo	Run command as administrator	sudo yum install python3 -y
yum / dnf	Package manager to install software	sudo yum update -y
mkdir	Create a new directory/folder	mkdir myapp
cd	Change directory (navigate folders)	cd myapp
nano	Text editor inside terminal	nano app.py
python3	Run Python programs	python3 --version
pip3	Install Python packages	sudo pip3 install flask
mysql	Connect to MySQL database	mysql -h endpoint -u admin -p

chmod	Change file permissions	chmod 400 key.pem
ls	List files in current directory	ls -la
cat	Show contents of a file	cat app.py
pwd	Show current directory path	pwd
ctrl+c	Stop a running program	Stop Flask server
ctrl+o	Save file in nano editor	Save app.py
ctrl+x	Exit nano editor	Exit after saving

Understanding Package Managers: yum and dnf

A package manager is a tool that automatically downloads and installs software on Linux. It is similar to the App Store on iPhone or Play Store on Android. Amazon Linux 2023 uses 'dnf' (Dandified YUM) as its package manager, and 'yum' also works as it is an alias to dnf. When you run 'sudo yum install python3 -y', the package manager connects to Amazon's software repository, downloads Python 3, and installs it automatically. The '-y' flag means 'yes to all questions' so the installation happens without asking for confirmation.

```
sudo yum update -y # Update all existing packages

sudo yum install python3 -y # Install Python 3

sudo dnf install mariadb105 -y # Install MariaDB (MySQL client)
```

Understanding the nano Text Editor

Nano is a simple text editor that runs inside the Linux terminal. Since EC2 has no graphical interface, we cannot use VS Code or Notepad to edit files on the server. Nano allows us to create and edit files directly in the terminal. When you type 'nano app.py', it opens the nano editor. You can type your code, then press Ctrl+O to save (O for Output/Write), press Enter to confirm the filename, and Ctrl+X to exit. The bottom of the nano screen always shows available keyboard shortcuts.

Key Combination	Action
Ctrl + O, then Enter	Save the file
Ctrl + X	Exit nano editor
Ctrl + K	Cut/Delete entire line
Ctrl + W	Search for text
Arrow Keys	Move cursor

Chapter 3 — AWS Services Explained

What is Amazon VPC (Virtual Private Cloud)?

Amazon VPC (Virtual Private Cloud) is a private, isolated section of the AWS cloud where you can launch AWS resources in a network that you define. Think of AWS as a massive shared city with millions of buildings. Without a VPC, your resources would be in a shared public space where anyone could potentially access them. A VPC is like building a private gated community inside that city — you own all the roads inside, you decide who can enter, and your resources are isolated from everyone else's.

When we created our VPC with CIDR block 10.0.0.0/16, we reserved a block of 65,536 private IP addresses (from 10.0.0.1 to 10.0.255.255) exclusively for our use. Every resource we created inside this VPC gets an IP address from this range. The VPC is the foundation of network security on AWS — every professional AWS project starts with creating a custom VPC.

- Subnets divide the VPC into smaller sections. Public subnets can receive internet traffic, private subnets cannot.
- Internet Gateway is the door between the VPC and the internet — like the main gate of the gated community.
- Route Tables are like traffic signs inside the VPC — they tell traffic which way to go.
- Security Groups are like bouncers at each resource — they decide what traffic is allowed in and out.

What is Amazon EC2 (Elastic Compute Cloud)?

Amazon EC2 (Elastic Compute Cloud) is a web service that provides resizable compute capacity in the cloud. In simple terms, EC2 lets you rent a virtual computer (server) in Amazon's data center. You can choose how powerful this computer is — how many CPUs, how much RAM, and how much storage it has. This virtual computer runs 24 hours a day, 7 days a week, without you having to worry about hardware failures, power, or cooling.

In this project, we launched a t2.micro EC2 instance running Amazon Linux 2023. The 't2' means it is a burstable general-purpose instance type, and 'micro' means it is the smallest size with 1 vCPU and 1 GB RAM. This is part of AWS Free Tier, meaning it is free for 750 hours per month for the first 12 months. We installed Python and Flask on this EC2 instance to create our backend web server that handles HTTP requests.

Key EC2 concepts used in this project: AMI (Amazon Machine Image) is the operating system template — we used Amazon Linux 2023. Key Pair is the SSH authentication method — we downloaded a .pem file to connect securely. Security Group acts as a virtual firewall controlling inbound and outbound traffic. We opened port 22 for SSH and port 80 for HTTP.

What is Amazon S3 (Simple Storage Service)?

Amazon S3 (Simple Storage Service) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Unlike EC2 which is a computer, S3 is purely storage — you store files (called objects) in containers (called buckets). S3 can store any type of file: images, videos, HTML files, PDF documents, backup files, or any other data.

One of S3's most powerful features is static website hosting. A static website is one that contains only HTML, CSS, and JavaScript files — no server-side processing needed. Since our frontend is just an HTML page, S3 can host it perfectly. When we enable static website hosting on an S3 bucket and make the files publicly accessible, S3 automatically serves those files to anyone who visits the website URL.

S3 is incredibly cheap and reliable — it stores data across multiple availability zones automatically, so there is virtually no chance of data loss. Major companies like Airbnb store billions of user photos in S3. Netflix stores their video files in S3. In our project, we hosted our HTML frontend on S3 at zero additional cost (within free tier limits of 5 GB storage and 20,000 GET requests per month).

What is Amazon RDS (Relational Database Service)?

Amazon RDS (Relational Database Service) is a managed relational database service that makes it easy to set up, operate, and scale a relational database in the cloud. A relational database stores data in tables with rows and columns — like a spreadsheet but much more powerful. It allows complex queries to find, filter, and join data across multiple tables.

Without RDS, you would have to manually install MySQL or PostgreSQL on an EC2 instance, configure it, set up backups, apply security patches, and manage performance — all very complex tasks. RDS handles all of this automatically. It provides automated backups, software patching, monitoring, and multi-AZ failover. You just create the database, connect to it, and start using it.

In this project, we used RDS with MySQL 8.4 engine on a db.t4g.micro instance (free tier). We placed it in private subnets, meaning it cannot be accessed directly from the internet — only from our EC2 instance inside the same VPC. This is a critical security practice: databases should never be exposed to the public internet.

Chapter 4 — Phase 1: Network Setup (VPC, Subnets, Internet Gateway, Route Table)

The first phase of any AWS project is always setting up the network. Before we can launch any servers or databases, we need to create the private network in which they will live. This is done using Amazon VPC. A well-designed network is critical for both security and performance. Many beginners skip this step and use the default VPC, but in professional projects, creating a custom VPC is always the right approach because it gives you full control over your network architecture.

Step 1 — Login to AWS Console and Set Region

The AWS Management Console is the web-based graphical interface for managing all AWS services. To access it, go to aws.amazon.com and click 'Sign In to the Console'. After logging in, the first thing to do is set the correct AWS region. AWS has data centers in many locations around the world — US, Europe, Asia Pacific, etc. Each location is called a region.

We selected Asia Pacific (Mumbai) — ap-south-1 — because it is geographically closest to Chennai, India. Choosing the nearest region gives the lowest latency (response time) for your users. All resources created in this project are in the Mumbai region.

■ RESULT: Logged in to AWS Console. Region set to Asia Pacific (Mumbai) ap-south-1.

Step 2 — Create Custom VPC

Navigate to VPC service using the search bar. In the VPC dashboard, click 'Create VPC' and select 'VPC Only' (not 'VPC and more', as we want to create subnets manually for learning purposes). Fill in the details as shown below and click Create VPC.

Field	Value	Explanation
Name tag	my-project-vpc	Name to identify this VPC
IPv4 CIDR	10.0.0.0/16	65,536 private IPs reserved for this VPC
IPv6	No IPv6	Not needed for this project
Tenancy	Default	Share physical hardware (cheaper)

The CIDR block 10.0.0.0/16 defines the IP address range for the entire VPC. The /16 prefix means the first 16 bits are fixed (10.0) and the remaining 16 bits are available for addresses — giving us 65,536 possible IP addresses from 10.0.0.0 to 10.0.255.255. We will carve smaller subnets out of this range.

■ RESULT: VPC 'my-project-vpc' created successfully. Status: Available.

Step 3 — Create 4 Subnets (2 Public + 2 Private)

Subnets are subdivisions of your VPC. We create two types: public subnets for resources that need internet access (EC2 web server) and private subnets for resources that should not be directly accessible from internet (RDS database). We create subnets in two Availability Zones (ap-south-1a and ap-south-1b) for high availability — if one AZ goes down, the other keeps running.

Go to VPC → Subnets → Create Subnet. Select your VPC and use 'Add new subnet' to add all four at once before clicking the final Create button.

Subnet Name	Type	Availability Zone	CIDR Block	Purpose
public-1a	Public	ap-south-1a	10.0.1.0/24	EC2 lives here
public-1b	Public	ap-south-1b	10.0.2.0/24	Backup public subnet
private-1a	Private	ap-south-1a	10.0.3.0/24	RDS lives here
private-1b	Private	ap-south-1b	10.0.4.0/24	Backup private subnet

Each subnet uses a /24 CIDR block which gives 256 IP addresses. For example, public-1a uses 10.0.1.0/24 which covers IPs from 10.0.1.0 to 10.0.1.255. AWS reserves 5 IPs in each subnet for internal use, leaving 251 usable IPs per subnet.

■ **RESULT: 4 subnets created. All showing status: Available.**

Step 4 — Create Internet Gateway and Attach to VPC

An Internet Gateway (IGW) is a horizontally scaled, redundant, and highly available VPC component that allows communication between your VPC and the internet. Without an Internet Gateway, resources inside your VPC — even in public subnets — cannot send or receive internet traffic. Think of it as the main entrance gate to your private community.

Go to VPC → Internet Gateways → Create Internet Gateway. Name it 'my-project-igw' and click Create. After creation, the IGW is in a 'Detached' state. You must attach it to your VPC by clicking Actions → Attach to VPC → Select 'my-project-vpc' → Attach Internet Gateway. One IGW can only be attached to one VPC at a time.

■ **RESULT: Internet Gateway 'my-project-igw' created and attached to my-project-vpc. State: Attached.**

Step 5 — Create Route Table for Public Subnets

A Route Table is a set of rules (called routes) that determine where network traffic from your subnet is directed. Every subnet in a VPC must be associated with a route table. By default, all subnets in a VPC can communicate with each other, but they cannot access the internet unless their route table has a route pointing to the Internet Gateway.

Go to VPC → Route Tables → Create Route Table. Name it 'public-rt' and select your VPC. After creation, click 'Edit routes' and add a new route: Destination = 0.0.0.0/0 (meaning all internet traffic), Target = your Internet Gateway. Then click 'Subnet Associations' → 'Edit subnet associations' and associate both public subnets (public-1a and public-1b). The private subnets do NOT get this route — they have no internet access by design.

Route	Destination	Target	Meaning
Local	10.0.0.0/16	local	Traffic within VPC stays inside
Internet	0.0.0.0/0	Internet Gateway	All other traffic goes to internet

■ **RESULT: Route table created with internet route. Associated with public-1a and public-1b subnets.**

Chapter 5 — Phase 2: EC2 Backend Server Setup

With our network ready, we can now launch the backend server. Amazon EC2 provides the computing power to run our Flask application. Flask is a lightweight Python web framework that makes it easy to build web applications and APIs. When a user visits our website, the request goes to the EC2 instance, Flask processes it, optionally queries the RDS database, and returns a response.

Step 6 — Launch EC2 Instance

Go to EC2 → Launch Instance. This wizard walks through all configuration options. We selected Amazon Linux 2023 as the AMI (Amazon Machine Image) — this is the operating system that will run on the virtual computer. Amazon Linux 2023 is optimized for AWS and comes with many AWS tools pre-installed.

The instance type is t2.micro which has 1 vCPU and 1 GB RAM — sufficient for a learning project. Key pair is critical: AWS uses public-key cryptography for SSH authentication. We created a new key pair, downloaded the .pem file, and saved it safely. This .pem file contains our private key — it can only be downloaded once. Losing it means losing SSH access to the instance forever.

Configuration	Value Selected	Reason
AMI	Amazon Linux 2023	Optimized for AWS, free tier eligible
Instance Type	t2.micro	Free tier, sufficient for learning project
Key Pair	backend server.pem	SSH authentication — saved to Downloads
VPC	my-project-vpc	Our custom private network
Subnet	public-1a	Public subnet — EC2 needs internet access
Public IP	Enabled	Needed so we can access from browser
Security Group	my-ec2-sg	Firewall rules for the instance

The Security Group 'my-ec2-sg' was configured with two inbound rules: SSH on port 22 to allow us to connect via terminal, and HTTP on port 80 to allow website visitors to access our Flask app. By default all outbound traffic is allowed so EC2 can download packages and make database connections.

■ RESULT: EC2 instance launched. Status: Running. Public IPv4: 13.232.140.70

Step 7 — Connect to EC2 via SSH

SSH (Secure Shell) is a cryptographic network protocol that allows secure remote login from one computer to another. When we connect via SSH, all communication between our laptop and the EC2 instance is encrypted. To connect, we need the EC2 public IP address (found in EC2 console), the key pair .pem file we downloaded, and the SSH command in PowerShell.

We used Windows PowerShell because the standard Command Prompt does not support the 'chmod' command used in Linux. PowerShell is a more powerful shell that supports SSH directly on Windows 10 and 11. The SSH command format is: ssh -i 'path-to-key.pem' username@server-ip. The username for Amazon Linux is always 'ec2-user'.

```
ssh -i "C:\Users\kalai\Downloads\backend server.pem" ec2-user@13.232.140.70
```

After running this command, SSH authenticates using our private key and establishes an encrypted connection. The terminal prompt changes to [ec2-user@ip-10-0-1-170 ~]$ which confirms we are now inside the EC2 server, thousands of kilometers away in Amazon's Mumbai data center.

Error Encountered	Cause	Solution Applied
chmod not recognized	Command Prompt doesn't support Linux commands	Switched to PowerShell
Connection timed out	SSH port 22 not open in security group	Added SSH inbound rule in security group
Identity file not accessible	Wrong .pem filename typed	Verified exact filename with dir *.pem command
Permission denied (publickey)	SSH source IP restriction	Changed security group SSH source to 0.0.0.0/0

■ RESULT: SSH connection successful. Terminal shows: [ec2-user@ip-10-0-1-170 ~]$

Step 8 — Install Python and Flask on EC2

Once connected to EC2, we need to install the software required to run our application. First, we update all existing packages to ensure the latest security patches are applied. Then we install Python 3 and pip3 (Python's package manager). Finally we install Flask using pip3 with sudo — it is important to use sudo here because we will run the app with sudo python3, and sudo uses root's Python environment.

```
sudo yum update -y # Update all system packages
```

```
sudo yum install python3 python3-pip -y # Install Python 3 and pip
```

```
sudo pip3 install flask # Install Flask as root user
```

```
python3 --version # Verify: should show Python 3.x.x
```

■ *Important: We must use 'sudo pip3 install flask' (not just 'pip3 install flask') because when we run 'sudo python3 app.py', it uses root's Python environment. If Flask is installed in the regular user's environment, root user cannot find it, causing ModuleNotFoundError.*

Error	Cause	Fix
ModuleNotFoundError: flask	Flask installed as regular user, run as root	Used: sudo pip3 install flask

■ RESULT: Python 3 and Flask installed successfully. Ready to write application code.

Step 9 — Create and Run Flask Application

Flask is a micro web framework for Python. It allows you to create web applications with very little code. A Flask application defines 'routes' — URL paths that trigger specific Python functions. When someone visits http://your-ip/, Flask calls the function decorated with @app.route('/') and returns whatever that function returns as the HTTP response.

We created a directory called 'myapp' to keep our project organized, then used the nano text editor to create app.py inside it. After writing the code, we saved with Ctrl+O and exited with Ctrl+X. Then we ran the application with sudo python3 because port 80 requires root privileges on Linux.

```
mkdir myapp # Create project directory

cd myapp # Enter the directory
```

```
nano app.py # Open nano text editor to write code
```

Code written inside nano:

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def home():

return "Hello! My AWS 3-Tier App is Running!"


if __name__ == '__main__':

app.run(host='0.0.0.0', port=80)
```

```
# Save: Ctrl+O, press Enter, then Ctrl+X to exit nano

sudo python3 app.py # Run Flask app on port 80
```

Setting host='0.0.0.0' is critical — by default Flask only listens on localhost (127.0.0.1) which means only the server itself can access it. Setting it to '0.0.0.0' makes Flask listen on all network interfaces, allowing external access via the EC2 public IP.

Error	Cause	Fix
IndentationError	Missing 4-space indent before return/app.run	Added 4 spaces before indented lines in na
ModuleNotFoundError: flask	Flask not installed for root user	Reinstalled with: sudo pip3 install flask

■ RESULT: Flask app running! Browser visit to http://13.232.140.70 showed: 'Hello! My AWS 3-Tier App is Running!'

Chapter 6 — Phase 3: RDS MySQL Database

The database is the memory of an application. Without a database, an application cannot store user information, process transactions, or remember anything between sessions. In this project, we use Amazon RDS with MySQL engine to store user data. RDS handles all database administration tasks automatically — backups, patching, monitoring — so we can focus on using the database rather than managing it.

Step 10 — Create RDS MySQL Instance

Go to RDS in the AWS Console. Click 'Create Database' and choose 'Standard Create' for full control over configuration. We selected MySQL 8.4 as the engine — MySQL is the world's most popular open-source relational database used by Facebook, Twitter, YouTube, and millions of other applications.

The Free Tier template automatically selects configurations that qualify for the AWS Free Tier (750 hours of db.t4g.micro per month). We placed the RDS instance in our custom VPC and selected 'Connect to an EC2 compute resource' which automatically configured the security group to allow our EC2 to connect to RDS. Setting Public Access to 'No' ensures the database endpoint is not reachable from the internet.

Setting	Value	Why
Engine	MySQL 8.4	Most popular relational database
Template	Free Tier	No cost for 750 hours/month
DB Identifier	myprojectdb	Unique name for this database instance
Username	admin	Master login username
Password	kalaiarasi123	Saved securely
Instance	db.t4g.micro	Free tier eligible, 2 vCPU, 1 GB RAM
VPC	my-project-vpc	Same VPC as EC2 for communication
Public Access	No	Database not exposed to internet
Availability Zone	No preference	AWS chooses for us

■ *After clicking Create Database, RDS takes 3-5 minutes to provision. Status changes from 'Creating' to 'Backing-up' to 'Available'. Do not proceed until status shows Available.*

■ **RESULT: RDS instance created. Endpoint: myprojectdb.cpwcqoq04ztq.ap-south-1.rds.amazonaws.com Port: 3306**

Step 11 — Install MySQL Client on EC2 and Connect

To connect to our RDS database from EC2, we need a MySQL client — a program that can communicate with MySQL databases. Amazon Linux 2023 does not include MySQL in its default repositories, but it includes MariaDB which is a fully compatible alternative. All MySQL commands work identically in MariaDB.

```
sudo dnf install mariadb105 -y # Install MySQL-compatible client
```

```
mysql -h myprojectdb.cpwcqoq04ztq.ap-south-1.rds.amazonaws.com -u admin
-pkalaiarasi123
```

The mysql command connects to the database. '-h' specifies the hostname (RDS endpoint), '-u' is the username, and '-p' followed immediately by the password logs in directly. The RDS endpoint is found in the RDS console under Connectivity & Security tab. After successful connection, you see the MySQL prompt.

Once connected to MySQL, we ran SQL (Structured Query Language) commands to create a database, create a table, insert data, and retrieve it. SQL is the standard language for interacting with relational databases.

```
CREATE DATABASE myapp;

USE myapp;

CREATE TABLE users (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100));

INSERT INTO users (name) VALUES ('Kalaiarasi');

SELECT * FROM users;
```

SQL Command	Meaning	Result
CREATE DATABASE myapp	Create a new database called myapp	Database created
USE myapp	Switch to myapp database	Database selected
CREATE TABLE users	Create users table with id and name columns	Table created
INSERT INTO users	Add a new row with name 'Kalaiarasi'	1 row inserted
SELECT * FROM users	Show all rows in users table	Displayed table with data

Error	Cause	Fix
No match for argument: mysql	MySQL not in Amazon Linux 2023 repos	Used: sudo dnf install mariadb105 -y
AUTO_INCREAMENT syntax error	Spelling mistake in SQL command	Corrected to AUTO_INCREMENT
Unknown database myapp	Database not created yet	Ran: CREATE DATABASE myapp; first

■ RESULT: Successfully connected to RDS. Table created and data inserted. SELECT query showed Kalaiarasi in users table.

Chapter 7 — Phase 4: S3 Static Website Frontend

The frontend is what users see when they visit a website. For this project, we created a simple HTML page and hosted it on Amazon S3. S3 is perfect for hosting static websites — it is extremely cheap (almost free for small sites), highly available (stores data across multiple locations), and automatically handles any amount of traffic without configuration.

Step 12 — Create S3 Bucket

An S3 bucket is a container for storing objects (files). Bucket names must be globally unique across all AWS accounts worldwide — no two S3 buckets in the world can have the same name. This is because S3 bucket URLs are publicly addressable like websites. We named our bucket 'my-frontend-kalaiarasi'.

When creating the bucket, we unchecked 'Block all public access'. By default, AWS blocks all public access to S3 buckets for security. Since we want to host a public website, we must disable this. AWS shows a warning and requires you to check an acknowledgement checkbox to confirm you understand the implications.

■ RESULT: S3 bucket 'my-frontend-kalaiarasi' created in ap-south-1 with public access enabled.

Step 13 — Enable Static Website Hosting

By default, S3 serves files as downloadable objects — clicking a URL downloads the file instead of displaying it in the browser. To make S3 serve files as a website, we must enable the Static Website Hosting feature. Go to the bucket → Properties tab → scroll to the bottom → Static website hosting → Edit.

Set Static website hosting to 'Enable', select 'Host a static website', and set Index document to 'index.html'. The index document is the default page that loads when someone visits the root URL. After saving, AWS generates a website endpoint URL that looks like: http://my-frontend-kalaiarasi.s3-website.ap-south-1.amazonaws.com

■ RESULT: Static website hosting enabled. Website endpoint URL generated in Properties tab.

Step 14 — Create HTML File, Upload, and Set Bucket Policy

We created an HTML file called index.html in VS Code on our local laptop. HTML (HyperText Markup Language) is the standard language for creating web pages. Our simple page displays a heading and some information about the project.

```
My AWS App

Welcome to My AWS 3-Tier Application!

Frontend: S3 | Backend: EC2 | Database: RDS

Built by Kalaiarasi
```

We uploaded this file to S3 by going to the bucket → Objects tab → Upload → Add files → select index.html → Upload. After uploading, we tried to visit the website URL but got an 'Access Denied' error. This happened because even though block public access is disabled at the bucket level, individual files are still private by default. We needed to add a Bucket Policy.

A Bucket Policy is a JSON document that defines permissions for the bucket and its contents. We added a policy that allows everyone (Principal: '*') to perform s3:GetObject (read files) on all objects in the bucket (Resource: arn:aws:s3:::my-frontend-kalaiarasi/*).

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Principal": "*",
"Action": "s3:GetObject",
"Resource": "arn:aws:s3:::my-frontend-kalaiarasi/*"
}]
}
```

Error	Cause	Fix
AccessDenied when visiting URL	No bucket policy to allow public reads	Added bucket policy with s3:GetObject permission

■ RESULT: Website live on S3! HTML page displayed correctly in browser. Frontend deployment complete.

Chapter 8 — Phase 5: GitHub Version Control

Git is a distributed version control system that tracks changes to files over time. It allows multiple developers to work on the same project simultaneously, keeps a complete history of all changes, and makes it easy to revert to previous versions if something goes wrong. GitHub is a web platform that hosts Git repositories and adds collaboration features like pull requests, issues, and project management tools.

For freelancers and job seekers, a GitHub profile is like a portfolio that shows real work. Clients and employers can visit your GitHub profile and see exactly what you have built, read your code, and assess your skills. Every project you complete should be pushed to GitHub with a clear README file explaining what it does.

Step 15 — Install Git, Create Repo and Push Code

Git was downloaded from git-scm.com/download/win and installed with default settings. After installation, we opened a fresh Command Prompt window (git commands only work after restarting the terminal) and configured our identity.

```
git config --global user.name "Kalaiarasi"

git config --global user.email "kalaiarasirajan73@gmail.com"
```

We created a new folder for the project, copied our index.html into it, then initialized a Git repository, added files to staging, committed, and pushed to GitHub. GitHub now requires Personal Access Tokens instead of passwords — this was generated from GitHub Settings → Developer Settings → Personal Access Tokens → Generate new token (classic) with 'repo' scope selected.

```
mkdir C:\Users\kalai\aws-3tier-app

cd C:\Users\kalai\aws-3tier-app

git init # Initialize git repository

git add . # Stage all files

git commit -m "AWS 3-tier web app project" # Create commit

git remote add origin https://github.com/Kalaiarasi73/aws-3tier-app.git

git push -u origin master # Push to GitHub
```

Error	Cause	Fix
401 Unauthorized	GitHub no longer accepts passwords	Used Personal Access Token (ghp_xxx) as passw
Repository not found	GitHub repo not created yet	Created new repo on github.com first

■ RESULT: Code pushed successfully to github.com/Kalaiarasi73/aws-3tier-app

Chapter 9 — All Errors Faced & Solutions

Encountering errors is a completely normal and essential part of learning. Every error teaches something new about how systems work. Below is a complete list of every error faced during this project and how it was solved. This section is very valuable — in job interviews and client discussions, being able to explain errors you faced and how you resolved them shows real experience.

#	Error	Phase	Root Cause	Solution
1	chmod not recognized	Phase 2	Windows CMD doesn't support Linux commands	Used PowerShell instead
2	Connection timed out (SSH)	Phase 2	Port 22 not open in security group	Added SSH inbound rule: port 22, 0.0.0.0/0
3	Identity file not accessible	Phase 2	Wrong filename typed in SSH command	Used dir *.pem to find exact filename
4	Permission denied (publickey)	Phase 2	SSH source IP restricted to old IP	Changed SSH source to Anywhere (0.0.0.0/0)
5	IP changed after restart	Phase 2	AWS assigns new public IP on restart	Got new IP from EC2 console
6	ModuleNotFoundError: flask	Phase 2	Flask installed as user, run as root	Used: sudo pip3 install flask
7	IndentationError in Flask	Phase 2	Missing 4-space indent in Python code	Added proper indentation in nano editor
8	sudo disabled error	Phase 2	Ran command in Windows laptop, not EC2	Connected to EC2 via SSH first
9	No match: mysql	Phase 3	MySQL not in Amazon Linux 2023 repos	Used sudo dnf install mariadb105 -y
10	AUTO_INCREAMENT error	Phase 3	Spelling mistake in SQL command	Corrected to AUTO_INCREMENT
11	Unknown database myapp	Phase 3	Database not created yet	Ran CREATE DATABASE myapp; first
12	Connection stuck/no password visible	Phase 3	Normal — Linux hides password typing	Just typed password and pressed Enter
13	AccessDenied on S3 URL	Phase 4	No bucket policy for public read	Added s3:GetObject bucket policy
14	401 Unauthorized on git push	Phase 5	GitHub no longer accepts passwords	Used Personal Access Token as password
15	Repository not found	Phase 5	GitHub repo not created before pushing	Created repo on github.com first

Chapter 10 — Project Summary & Key Learnings

Complete Task Checklist

Phase	Task Completed	Status
Phase 1 — Network	Created custom VPC with CIDR 10.0.0.0/16	■ Done
Phase 1 — Network	Created 4 subnets (2 public, 2 private) in 2 AZs	■ Done
Phase 1 — Network	Created Internet Gateway and attached to VPC	■ Done
Phase 1 — Network	Created route table with internet route for public subnets	■ Done
Phase 2 — EC2	Launched t2.micro EC2 in public subnet	■ Done
Phase 2 — EC2	Configured security group with SSH and HTTP rules	■ Done
Phase 2 — EC2	Connected to EC2 via SSH from Windows PowerShell	■ Done
Phase 2 — EC2	Installed Python 3 and Flask on EC2	■ Done
Phase 2 — EC2	Created Flask app with nano text editor	■ Done
Phase 2 — EC2	Flask app running and accessible via browser	■ Done
Phase 3 — RDS	Created RDS MySQL db.t4g.micro in private subnet	■ Done
Phase 3 — RDS	Installed MariaDB client on EC2	■ Done
Phase 3 — RDS	Connected EC2 to RDS via MySQL client	■ Done
Phase 3 — RDS	Created database, table, inserted and queried data	■ Done
Phase 4 — S3	Created S3 bucket with public access enabled	■ Done
Phase 4 — S3	Enabled static website hosting with index.html	■ Done
Phase 4 — S3	Uploaded HTML file and added public bucket policy	■ Done
Phase 4 — S3	Website accessible via S3 website URL	■ Done
Phase 5 — GitHub	Installed Git and configured identity	■ Done
Phase 5 — GitHub	Created GitHub repository aws-3tier-app	■ Done
Phase 5 — GitHub	Pushed code using Personal Access Token	■ Done

Key Learnings from This Project

1. VPC is the foundation of AWS security

Never use the default VPC for real projects. Always create a custom VPC so you have full control over your network architecture, IP addressing, and security boundaries.

2. Private subnets protect databases

Databases should never be in public subnets. By placing RDS in a private subnet with no internet route, we ensure only authorized resources (EC2) can access it.

3. Linux sudo is essential for servers

Most server operations require sudo (admin) privileges. Understanding when to use sudo and why is a fundamental Linux skill for cloud engineers.

4. Package names differ by Linux distribution

Amazon Linux 2023 uses 'mariadb105' instead of 'mysql'. Always check the correct package names for your specific Linux distribution.

5. Python environment matters (sudo vs user)

When you run sudo python3, it uses root's Python environment. Always install packages with sudo pip3 if you plan to run the app with sudo python3.

6. S3 public access has two layers

Disabling block public access at bucket level is not enough — you also need a bucket policy to actually allow public reads. Both settings must be configured.

7. GitHub requires Personal Access Tokens

Since August 2021, GitHub no longer accepts account passwords for git push. Personal Access Tokens are required and are more secure.

8. Security Groups are stateful firewalls

Security Groups remember connection state — if you allow inbound SSH, the response traffic is automatically allowed outbound. Only define what you want to allow in.

■ Project 1 of 5 Successfully Completed! ■

GitHub: github.com/Kalaiarasi73/aws-3tier-app | Next: Project 2 — CI/CD Pipeline with Jenkins