

# Airline Ticket Booking System – AWS Cloud Project

## AWS Services You Will Use

- **Amazon S3** → Host the website (UI)
- **Amazon API Gateway** → Expose APIs
- **AWS Lambda** → Business logic (search & booking)
- **Amazon DynamoDB** → Store flight & booking info
- **Amazon SES** → Send confirmation emails
- **IAM** → Permissions & security
- **SES** – Email Service
- **CloudWatch** – log Groups

### → Step 1: Setup Flight Booking Website

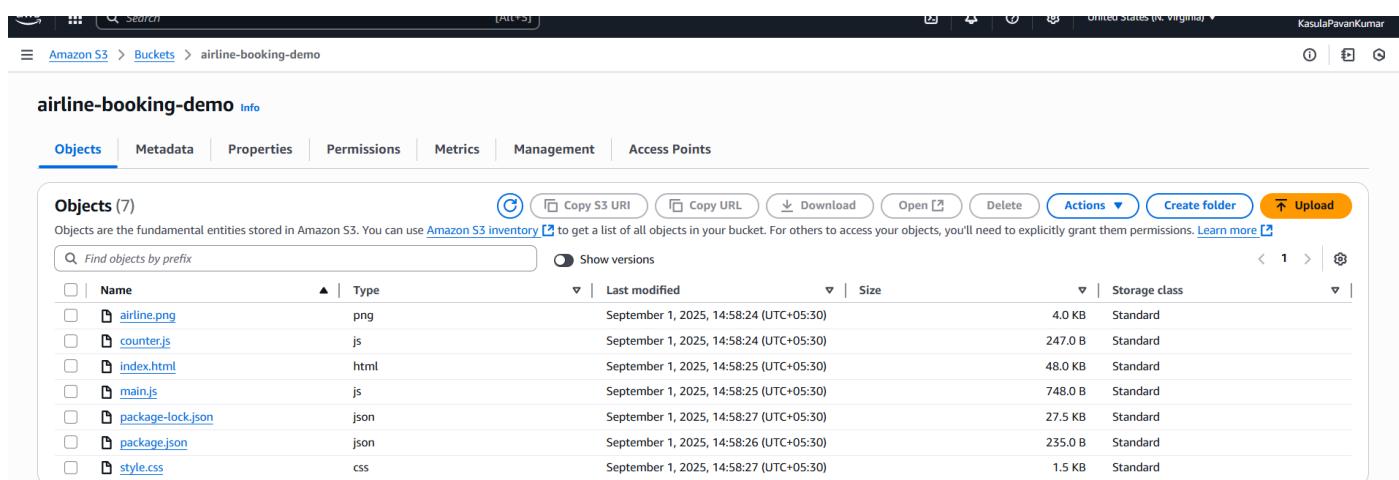
- Create an S3 Bucket
- Enable Static Website Hosting
- Upload Front-End Files
- Make Files Public
- Test Website



Uploading  
Total remaining: 184 files: 18.1 MB (98.62%)  
Estimated time remaining: 19 minutes  
Transfer rate: 16.2 KB/s

Name	Folder	Type	Size	Status	Error
.gitignore	AirLineWebsite/	text/plain	253.0 B	Succeeded	-
booking.js	AirLineWebsite/	text/javascript	7.9 KB	Succeeded	-
flights.js	AirLineWebsite/	text/javascript	5.1 KB	Succeeded	-
index.html	AirLineWebsite/	text/html	7.1 KB	Succeeded	-

S3 creation Image(1a)



Objects (7)  
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](#)

Name	Type	Last modified	Size	Storage class
airline.png	png	September 1, 2025, 14:58:24 (UTC+05:30)	4.0 KB	Standard
counter.js	js	September 1, 2025, 14:58:24 (UTC+05:30)	247.0 B	Standard
index.html	html	September 1, 2025, 14:58:25 (UTC+05:30)	48.0 KB	Standard
main.js	js	September 1, 2025, 14:58:25 (UTC+05:30)	748.0 B	Standard
package-lock.json	json	September 1, 2025, 14:58:27 (UTC+05:30)	27.5 KB	Standard
package.json	json	September 1, 2025, 14:58:26 (UTC+05:30)	235.0 B	Standard
style.css	css	September 1, 2025, 14:58:27 (UTC+05:30)	1.5 KB	Standard

S3 creation Image(1b)

## → Step 2: Create DynamoDB Tables

- Create Flights Table
- Add Sample Flight Data (after table is created)
- Create Bookings Table

The screenshot shows the AWS DynamoDB Tables page. On the left, there's a sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a DAX section. The main area is titled "Tables (2)" and lists two tables: "Bookings" and "Flights". Both tables are active, have a partition key of type String, and a sort key of type String. They both have 0 indexes, 0 replication regions, and deletion protection off. The Bookings table has 1 item and the Flights table has 2 items. There are buttons for Actions, Delete, and Create table.

DynamoDB creation Image(1a)

This screenshot shows the "Explore items" section for the "Flights" table. The sidebar is identical to the previous one. The main area is titled "Flights" and includes a "Scan or query items" section with "Scan" selected. It also shows a table header for "Flight" with columns: flightID, availableSeats, date, destination, price, and source. Two flights are listed: AI204 (90 seats, Bangalore, 5500, Delhi) and AI203 (42 seats, 2025-09-05, Delhi, 4500, Mumbai). A success message at the bottom says "Completed - Items returned: 2 · Items scanned: 2 · Efficiency: 100% · RCU consumed: 2". There are buttons for Actions, View table details, and Create item.

DynamoDB creation Image(1b)

This screenshot shows the "Explore items" section for the "Bookings" table. The sidebar is identical. The main area is titled "Bookings" and includes a "Filters - optional" section with "Run" and "Reset" buttons. A warning message says "Stopped - Items returned: 50 · Items scanned: 50 · Efficiency: 100% · RCU consumed: 1.5". Below is a table titled "Table: Bookings - Items returned (50)" with columns: bookingID, email, flightID, status, and userName. The table lists 50 entries, each with a bookingID starting with 6a199930-0e6c-4e9b..., email like pavankuma..., flightID like AI203, status like CONFIRMED, and userName like Pavan. There are buttons for Actions, Create item, and Retrieve next page.

DynamoDB creation Image(1c)

### → Step 3: Create Lambda Functions

We need two **serverless backend functions**:

- `searchFlights` → Fetch available flights from DynamoDB.
- `bookTicket` → Store booking info + reduce available seats + send confirmation later.

**Create Lambda Function – `searchFlights`**(Permissions → Create a new role with **DynamoDB read-only access**)

**Create Lambda Function – `bookTicket`**(Permissions → Role with **DynamoDB full access + SES**)

Runtime → **Python 3.12** (or Our Wish Based on Knowledge about project -SP).

**Book Ticket code (Lambda) –**

```
import json
import boto3
import uuid

dynamodb = boto3.resource('dynamodb')
flights_table = dynamodb.Table('Flights')
bookings_table = dynamodb.Table('Bookings')
ses = boto3.client('ses')

def lambda_handler(event, context):
    try:
        # Parse body from API Gateway
        flight_id = event.get("flightID")
        user_name = event.get("userName")
        email = event.get("email")

        if not flight_id or not user_name or not email:
            return {
                "statusCode": 400,
                "headers": {"Access-Control-Allow-Origin": "*"},
                "body": json.dumps({"message": "Missing required fields"})
            }
    except Exception as e:
        print(f"Error: {e}")
        return {
            "statusCode": 500,
            "body": json.dumps({"message": "Internal Server Error"})
        }
```

```

# Generate booking ID

booking_id = str(uuid.uuid4())


# Save booking

bookings_table.put_item(
    Item={
        "bookingID": booking_id,
        "flightID": flight_id,
        "userName": user_name,
        "email": email,
        "status": "CONFIRMED"
    }
)

# Decrease available seats (ensure not below zero)

flights_table.update_item(
    Key={"flightID": flight_id},
    UpdateExpression="SET availableSeats = if_not_exists(availableSeats, :start) - :val",
    ExpressionAttributeValues={":val": 1, ":start": 100}, # default 100 seats if missing
    ConditionExpression="availableSeats >= :val"
)

# Send confirmation email

ses.send_email(
    Source="pavankumarkasula73@gmail.com", # must be verified in SES
    Destination={"ToAddresses": [email]},
    Message={
        "Subject": {"Data": "Flight Booking Confirmation"},
        "Body": {
            "Text": {

```

```

        "Data": f"Hello {user_name},\n\nYour booking for flight {flight_id} is CONFIRMED.\nBooking
ID: {booking_id}\n\nThank you for choosing our Airline!"

    }

}

)

return {
    "statusCode": 200,
    "headers": {"Access-Control-Allow-Origin": "*"},
    "body": json.dumps({
        "message": "Booking confirmed & email sent",
        "bookingID": booking_id
    })
}

```

except Exception as e:

```

print("Error:", str(e))

return {
    "statusCode": 500,
    "headers": {"Access-Control-Allow-Origin": "*"},
    "body": json.dumps({"message": "Internal server error", "error": str(e)})
}

```

### Search Flights Code (Lambda) :

```

import json

def lambda_handler(event, context):
    # Get the 'body' field
    body = event.get('body', '{}')

```

```
# If 'body' is already a dict, use it directly.
```

```
# If it's a string, parse it to dict.
```

```
if isinstance(body, dict):
```

```
    data = body
```

```
else:
```

```
    data = json.loads(body)
```

```
# Now you can safely access keys
```

```
source = data.get('source')
```

```
destination = data.get('destination')
```

```
date = data.get('date')
```

```
# Your logic here...
```

```
return {
```

```
    'statusCode': 200,
```

```
    'body': json.dumps({
```

```
        'source': source,
```

```
        'destination': destination,
```

```
        'date': date
```

```
    })
```

```
}
```

Functions (2)					
<input type="text"/> Filter by attributes or search by keyword					
<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	<a href="#">searchFlights</a>	-	Zip	Python 3.12	2 days ago
<input type="checkbox"/>	<a href="#">bookTicket</a>	-	Zip	Python 3.12	26 minutes ago

Lambda function image(1a)

The screenshot shows the AWS API Gateway Resources page. On the left, the navigation bar includes 'API Gateway' > 'APIs' > 'Resources - AirlineBookingAPI (gcglletwn9)'. The main panel displays a 'Resources' section with a 'Create resource' button. A resource named '/' is listed, containing two methods: '/bookTicket' (OPTIONS, POST) and '/searchFlights' (OPTIONS, POST). To the right, 'Resource details' show the path '/' and Resource ID 'wcrahp84mi'. The 'Methods (0)' section indicates 'No methods defined.' Buttons for 'Update documentation', 'Enable CORS', 'Delete', and 'Create method' are present.

Lambda function image(1b)

The screenshot shows the AWS IAM Roles page. The top banner says 'Policies have been successfully attached to role.' The main section is titled 'Roles (15) Info' with a description of what IAM roles are. It lists three roles: 'ChatbotLambda-role-x0ey7x7c' (AWS Service: lambda), 'rds-monitoring-role' (AWS Service: monitoring.rds), and 'searchFlights-role-cdoujpjv' (AWS Service: lambda). A search bar and a 'Create role' button are also visible.

Lambda function image(1c)

## Step 4: Create APIs with Amazon API Gateway

- Here we connect your frontend (S3 website) to the backend (Lambda functions).
- **Open API Gateway -> Choose REST API**
- Create Resource for Flights Search -> selected → Click Actions → Create Method → Choose POST
- Create Resource for Booking Tickets -> selected → Click Actions → Create Method → Choose POST
- Enable CORS (Frontend Access)
- Finally we Deploy our API here.(copy & paste your api endpoints in notepad for future use case.)
- After we test API'S

Search Flights Api test code –

```
{
  "source": "Mumbai",
  "destination": "Delhi",
  "date": "2025-09-05"
}
```

- After we test API'S

Ticket Booking Api test code –

```
{
  "flightID": "AI203",
  "userName": "Pavan",
  "email": "pavan@example.com"
}
```

The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar for 'API Gateway' with sections like 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', and 'API: AirlineBookingAPI' which has 'Resources', 'Stages' (selected), 'Authorizers', 'Gateway responses', 'Models', 'Resource policy', 'Documentation', 'Dashboard', and 'API settings'. The main area is titled 'Stages' and shows a table for the 'Dev' stage. The 'Stage details' section includes fields for 'Stage name' (Dev), 'Rate Info' (10000), 'Cache cluster Info' (Inactive), 'Default method-level caching' (Inactive), and 'Web ACL' (-). The 'Logs and tracing' section includes 'CloudWatch logs' (Inactive), 'Detailed metrics' (Inactive), and 'Data tracing' (Inactive). A tooltip 'Copied' is shown over the URL field, which contains 'https://goglletwn9.execute-api.us-east-1.amazonaws.com/Dev'. Below the table, it says 'Active deployment' with the timestamp 'htqOn August 30, 2025, 09:09 (UTC+05:30)'. There are 'Stage actions' and 'Edit' buttons at the top right.

Api Gateway Image(1a)

The screenshot shows the 'Testing(post)' results for the '/searchFlights' endpoint. It includes sections for 'Request' (method POST, path '/searchFlights'), 'Latency ms' (34), 'Status' (200), 'Response body' ({"statusCode": 200, "body": "{\"source\": \"Mumbai\", \"destination\": \"Delhi\", \"date\": \"2025-09-05\""}}, 'Response headers' (Content-Type: application/json, Access-Control-Allow-Origin: \*, X-Amzn-Trace-Id: Root=1-68b2873b-2bd0a00bbcfd4e4cb9e3a877;Parent=440cb2721c451492;Sampled=0;Lineage=1:795844f9:0), and 'Logs' (empty).

Api Gateway Testing(post) Image(1a)

### → Step 5(a): Connect Frontend with APIs.

- Get API Gateway URL
- We'll use this in frontend code for connection between them
- Update URL in required fields in frontend.
- Upload Updated Files to S3
- Refresh our website URL in S3 and check.
- After Booking Ticket Booking we need to get booking details to used Gmail.

### → Step 5(b): Email Confirmation with Amazon SES.

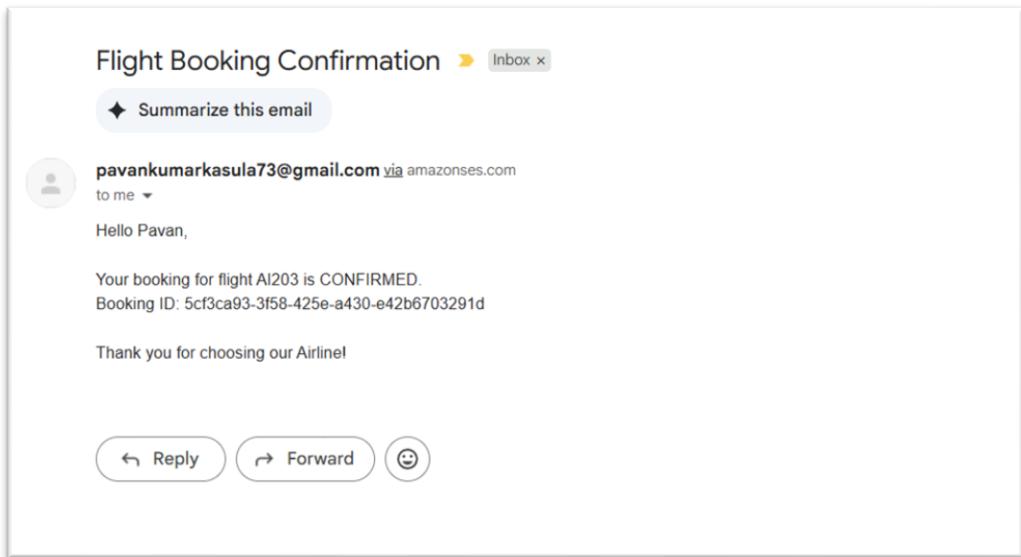
- In Verified identities → Create identity
- Choose **Email address**.
- Enter your email (e.g., [pavankumarkasula73@gmail.com](mailto:pavankumarkasula73@gmail.com)).
- You'll get a verification email → Click the link to confirm.
- **Add SES Permission to Lambda -> Your bookTicket Lambda role**
- Attach policy → AmazonSESFullAccess

The screenshot shows the homepage of the PA1wings website. At the top, there's a green header bar with the PA1wings logo and navigation links for Home, Flights, Destinations, and Contact. Below the header, a progress bar indicates four steps: '1' (selected), '2', '3', and '4'. A large green banner at the top says 'Welcome to PA1wings' and 'Your trusted partner for seamless domestic travel across India'. Below the banner, the main heading is 'Find Your Perfect Flight'. There are four input fields: 'From' (Select departure city), 'To' (Select destination), 'Departure Date' (mm/dd/yyyy), and 'Passengers' (1 Passenger). A green 'Search Flights' button is centered below these fields. The background of the page is light green.

Hosted Website(S3/EC2) Image(1a)

The screenshot shows a confirmation page for a flight booking. At the top, a green circular icon with a checkmark is followed by the text 'Booking Confirmed!'. Below it, a message states 'Your flight has been successfully booked. A confirmation email has been sent to your email address.' The confirmation number is 'Confirmation #PA3LP212F00'. Under 'Flight Details', it lists: Flight: SW309, Route: DEL → HYD, Date: 2025-09-30, Seats: 14F. Under 'Passengers', it lists: Kasula PavanKumar - Seat 14F. The total amount paid is 'Total Paid: ₹7499'. At the bottom, there are two buttons: 'Print Ticket' (green) and 'Book Another Flight' (dark grey).

Hosted Website (Booking Completed) Image(1b)



**Hosted Website (MAIL CONFIRMATION) Image(1c)**

Check CloudWatch Log metrics for Log analysis for Errors.

Log group	Log class	Anomaly detection	Data protection	Sensitive data count	Retention	Metric filters	Transformer
/aws/lambda/AWSDynamoDb01	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/bookTicket	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/searchFlights	Standard	Configure	-	-	Never expire	-	-
/aws/lex/WUN5FOFMZA	Standard	Configure	-	-	Never expire	-	-

**Cloudwatch(Logs)(1a)**

Log stream	Last event time
2025/09/01/[LATEST]aea0379c557d405e977782dd574cb49a	2025-09-01 10:14:49 (UTC)
2025/09/01/[LATEST]fe895f8b397942a7821199539647578f	2025-09-01 09:57:23 (UTC)
2025/09/01/[LATEST]6e70ea62d85c4f3cb0f822274cfbcff	2025-09-01 09:53:25 (UTC)

**Cloudwatch(Logs)(1b)**

The screenshot shows the AWS CloudWatch Log groups interface. On the left, there's a navigation sidebar with options like CloudWatch, Favorites and recents, Dashboards, AI Operations, Alarms, Logs (Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights), Metrics, Application Signals (APM), GenAI Observability (Preview), Network Monitoring, and Insights (Settings, Telemetry config, Getting Started). The main area is titled 'Log group details' for '/aws/lambda/AWSDynoDb01'. It displays various metrics and settings: Log class (Info, Standard), ARN (arn:aws:logs:us-east-1:747804225529:log-group:/aws/lambda/AWSDynoDb01:\*), Creation time (18 days ago), Retention (Never expire), and Stored bytes (.10.11.KB). It also shows Metric filters (0), Subscription filters (0), Contributor Insights rules (-), KMS key ID (-), and Anomaly detection (Configure). On the right, there are sections for Data protection (Sensitive data count -), Custom field indexes (Configure), and Transformer (Configure). Below this, the 'Log streams' tab is selected, showing three log streams: '2025/08/14/[\$LATEST]fb39b9ed94cd47179a88382e68db8718' (Last event time: 2025-08-14 05:15:57 UTC), '2025/08/13/[\$LATEST]82e755cf669840749c70a68c1cfa486f' (Last event time: 2025-08-14 05:09:54 UTC), and '2025/08/13/[\$LATEST]5ab8df890cac4245a4ca42892e6a8757' (Last event time: 2025-08-14 05:05:04 UTC). There are buttons for Create log stream and Search all log streams.

### Cloudwatch(Logs)(1c)

## → STEP – 6(OPTIONAL)

In case of trouble with S3 bucket static website, we can create an EC2 machine and clone the repo there, and install required packages in machine , After check with machine ip and 8000.

### EC2 MACHINE : CONTAINS WEBSITE( UPDATED VIA GIT CLONE)

http-server -p 8000

The screenshot shows the AWS EC2 Instances interface. The left sidebar has options for EC2 (Dashboard, EC2 Global View, Events), Instances (Instances), and a search bar. The main area shows 'Instances (1/1) Info' for an instance named 'AirlineMachine' (i-0489028784f75a005). The instance is running (t3.micro), has 3/3 checks passed, and is in the us-east-1a availability zone. A public IP of 54.161.32.132 is listed. At the bottom, there's a terminal window showing the output of the 'http-server -p 8000' command. The output includes configuration details like AutoIndex: visible, CORS: disabled, and connection timeout, followed by logs of requests from Mozilla Firefox and Mozilla/5.0 clients.

```

AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://127.0.0.1:8000
  http://10.0.1.252:8000
Hit CTRL-C to stop the server

^Chttp-server stopped.
root@ip-10-0-1-252:/home/ubuntu/PalAirlines/Airlinewebsite# http-server -p 8000
Starting up http-server, serving .
http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://127.0.0.1:8000
  http://10.0.1.252:8000
Hit CTRL-C to stop the server

[2025-09-01T09:51:26.160Z] "GET /" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0"
(node:15092) [DEP0066] DeprecationWarning: OutgoingMessage.prototype._headers is deprecated
(Use `node --trace-deprecation ...` to show where the warning was created)
[2025-09-01T10:05:11.023Z] "GET /airline.png" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0"

```

#### → LEARNINGS & CHALLENGES FROM THESE PROJECTS:

- Gained practical experience with S3 (static website hosting), DynamoDB (NoSQL database), API Gateway (REST API management), Lambda (serverless functions), and SES (email notifications).
- Learned how these services interact to form a complete cloud-based application.
- Understood how to build applications without managing servers using AWS Lambda.
- Learned the importance of IAM roles & permissions to securely allow services to talk to each other.
- Learned how browser security policies (CORS) affect API calls and how to configure API Gateway headers to make frontend and backend communicate smoothly.
- Explored DynamoDB partition keys and schema design for handling **Flights** and **Bookings** efficiently.
- Integrated SES to send booking confirmation emails, which introduced me to **event-driven notifications**.

#### → CHALLENGES:

- Initially, My Lambda function failed because request payloads come wrapped inside event["body"]
- The frontend fetch () calls were blocked due to missing CORS headers.
- Fixed this by enabling CORS in API Gateway and adding headers in Lambda responses.
- Logs in CloudWatch and API Gateway execution logs were critical to identify why Lambda was returning 400 Bad Request.
- After booking a ticket, I noticed that no confirmation email was received.

This created a lot of confusion since the Lambda executed successfully, but the user never got the expected notification.

Guidance received from Sir / Madam:

My guide suggested two possible troubleshooting paths:

- Deploy the frontend on **EC2** instead of S3 to rule out any static hosting restrictions.
- Double-check whether the **API Gateway invoke URL** was correctly configured in the frontend fetch () code.
- Learned to trace problems step by step using logs and by testing services independently (e.g., calling API Gateway directly via Postman).
- Gained awareness that debugging in cloud projects isn't just about fixing code—it also requires validating **infrastructure setup and integrations**.
- Learned to rely on **logs, monitoring, and guided troubleshooting** rather than guessing.
- Understood that sometimes **workarounds** (like deploying on EC2) help isolate issues, even if the root cause is elsewhere.