Lab 04: Contact Form + DynamoDB

Lab Overview

Extend the Lab 03 contact form to also store every submission in an Amazon DynamoDB table. Each form entry is emailed via SES AND permanently recorded in the database. This is a core serverless data pipeline pattern: API Gateway → Lambda → SES + DynamoDB.

◆ **PREREQ:** Complete Lab 03 (Serverless Contact Form) before this lab. You will update the Lambda function and IAM role from Lab 03.

Service	Purpose	Free Tier
Amazon DynamoDB	Fully managed NoSQL database — stores every contact form submissio	25 GB + 25 WCU/RCU free foreve
AWS Lambda	Updated function that writes to DynamoDB AND sends email via SES	1M requests/mo free
Amazon API Gateway	HTTPS endpoint — unchanged from Lab 03	1M calls/mo free
Amazon SES	Sends email notification — unchanged from Lab 03	62,000 emails/mo free
AWS IAM	Updated role granting Lambda permission to write to DynamoDB	Always free

■ **NOTE:** DynamoDB Free Tier: 25 GB storage + 25 read/write capacity units per month — permanently free, no time limit.

1 Amazon DynamoDB
Create the ContactFormSubmissions Table

Create the table that will store all contact form submissions.

1. Search for DynamoDB → click Tables → Create table
2. Table name: ContactFormSubmissions
3. Partition key: submissionId → Type: String
4. Table settings: Customize settings → Capacity: On-demand
5. Click Create table → wait for status: Active

✓ **TIP:** On-demand capacity means you pay only per request. At contact form volumes this is effectively free.

2 AWS IAM
Update the Lambda IAM Role

1. IAM → Roles → search for LambdaSESRole → click it
2. Permissions tab → Add permissions → Attach policies
3. Search AmazonDynamoDBFullAccess → check it → Add permissions

■ **NOTE:** For production use a custom policy granting only dynamodb:PutItem on the specific table ARN.

3 AWS Lambda
Replace the Function Code

Open ContactFormHandler → Code tab → delete all existing code and paste the following. Replace both email addresses.

```
import json, boto3, uuid
from datetime import datetime, timezone

ses      = boto3.client('ses', region_name='us-east-1')
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table    = dynamodb.Table('ContactFormSubmissions')

SENDER_EMAIL    = 'your-verified-email@example.com'  # replace
RECIPIENT_EMAIL = 'your-verified-email@example.com'  # replace

def lambda_handler(event, context):
    try:
        body    = json.loads(event.get('body', '{}'))
        name    = body.get('name', 'Unknown')
        email   = body.get('email', 'Unknown')
        message = body.get('message', 'No message')
        sub_id  = str(uuid.uuid4())
        ts      = datetime.now(timezone.utc).isoformat()

        # 1 - Write to DynamoDB
        table.put_item(Item={
            'submissionId': sub_id,
            'timestamp':    ts,
            'name':         name,
            'email':        email,
            'message':      message
        })

        # 2 - Send email via SES
        ses.send_email(
            Source=SENDER_EMAIL,
            Destination={'ToAddresses': [RECIPIENT_EMAIL]},
            Message={
                'Subject': {'Data': f'New contact from {name}'},
                'Body': {'Text': {'Data':
                    f'Name: {name}\nEmail: {email}\n'
                    f'Message: {message}\n\nID: {sub_id}\nTime: {ts}'
                }}
            }
        )
        return {'statusCode': 200,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps({'message': 'Saved!', 'submissionId': sub_id})}
    except Exception as e:
        print(f'Error: {str(e)}')
        return {'statusCode': 500,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps({'error': str(e)})}
```

■ **WARNING:** Replace both 'your-verified-email@example.com' with your SES verified address before clicking Deploy.

1. Click Deploy → wait for Changes deployed banner

4 AWS Lambda
Test the Updated Function

1. Click Test tab → use or create TestContact event with this JSON:

```
{"body": "{\"name\": \"Test User\", \"email\": \"test@example.com\", \"message\": \"DynamoDB test\"}", "httpMethod": "POST"
```

2. Click Test → confirm Execution result: succeeded
3. Response body should include a submissionId (UUID string)
4. Check your inbox — email should arrive with the submission ID

✓ **TIP:** The submissionId in the response is the exact DynamoDB partition key for this record.

5 Amazon DynamoDB View Submissions in the Console

1. DynamoDB → Tables → ContactFormSubmissions
2. Click Explore table items (orange button, top right)
3. Your test submission appears as an item in the table
4. Click it to expand and see all 5 attributes: submissionId, timestamp, name, email, message
5. Submit the live contact form on your website — refresh the table — new item appears instantly

■ **NOTE:** Every real contact form submission from your website is now permanently stored in DynamoDB. You have a complete record of every person who contacts you.

6 Amazon DynamoDB Scan and Query Your Data

1. On the Explore table items page click Scan / Query
2. Scan (default) → Run — returns all items in the table
3. Switch to Query → enter a submissionId value → Run — returns only that item

✓ **TIP:** Scan reads every item (slow at scale). Query uses the partition key (fast at any scale). In production add a Global Secondary Index on email or timestamp for efficient filtering.

Verification Checklist

- ■ DynamoDB table ContactFormSubmissions created with On-demand capacity and Active status
- ■ LambdaSESRole updated with AmazonDynamoDBFullAccess policy
- ■ Lambda code replaced with updated version — both email addresses updated
- ■ Lambda deployed — Changes deployed banner confirmed
- ■ Test event returns succeeded with submissionId in response body
- ■ Email received with submission ID included in message body
- ■ DynamoDB Explore table items shows the test submission
- ■ All 5 attributes visible: submissionId, timestamp, name, email, message
- ■ Live contact form submission appears in DynamoDB table

What You Learned

- Amazon DynamoDB — fully managed NoSQL database, On-demand capacity, tables, items, and attributes
- PutItem — how Lambda writes records to DynamoDB using the boto3 DynamoDB resource
- UUID generation — creating unique partition keys for every database record
- Multi-service Lambda — a single function writing to DynamoDB and calling SES simultaneously

- IAM policy updates — adding DynamoDB permissions to an existing role
- Scan vs Query — full table reads vs key-based lookups in DynamoDB
- Serverless data pipeline — API Gateway → Lambda → DynamoDB + SES is a core production pattern

---

Lab Cleanup

✗ **IMPORTANT:** Delete these resources when finished.

#	Resource	How to Delete
1	DynamoDB Table	DynamoDB → Tables → ContactFormSubmissions → Delete → confirm
2	IAM Policy	IAM → Roles → LambdaSESRole → detach AmazonDynamoDBFullAccess
3	Lambda Code	Optionally revert ContactFormHandler to the Lab 03 version

■ **NOTE:** The API Gateway endpoint does not need to be deleted — only the DynamoDB table and IAM policy are specific to this lab.