Lab 08: Email Verification & User Registration

Lab Overview

Build a two-step email verification and user registration flow. A user submits a form, receives a verification email, clicks the link, and only then is their record permanently saved to DynamoDB and the site owner notified. Unverified submissions auto-expire after 24 hours via DynamoDB TTL.

◆ **PREREQ:** Complete Lab 03 (Serverless Contact Form) and Lab 04 (DynamoDB) before this lab.

Service	Purpose	Free Tier
Amazon SES	Sends the verification email to the user and the notification to the owner	62,000 emails/mo free
Amazon DynamoDB	Stores PENDING registrations with TTL and CONFIRMED registrations permanently	25 GB free forever
AWS Lambda	Two functions: RegistrationHandler (start) and ConfirmationHandler (confirm)	1M requests/mo free
Amazon API Gateway	Two routes: POST /register and GET /confirm	1M calls/mo free
Amazon S3	Hosts register.html and confirmed.html	Free Tier
AWS IAM	Role granting Lambda permission to use SES and DynamoDB	Always free

■ **NOTE:** DynamoDB TTL automatically deletes PENDING records that were never confirmed after 24 hours. No scheduled jobs or manual cleanup needed.

Architecture — Two-Step Flow

Step	What Happens	AWS Service
1	User submits name + email on register.html	S3 / Browser
2	JavaScript POSTs to API Gateway POST /register	API Gateway
3	Lambda generates a UUID verification token	Lambda
4	Lambda writes PENDING record to DynamoDB with 24-hour TTL	DynamoDB
5	Lambda sends a verification email to the user via SES	SES
6	User clicks the Verify My Email button in the email	Email Client
7	Browser GETs API Gateway GET /confirm?token=UUID	API Gateway
8	Lambda looks up the token in DynamoDB	Lambda
9	Lambda marks the record as CONFIRMED and removes TTL	DynamoDB
10	Lambda sends an owner notification email via SES	SES
11	Lambda redirects the browser to confirmed.html	S3 / Browser

1 **Amazon DynamoDB**
Create the UserRegistrations Table

1. DynamoDB → Tables → Create table
2. Table name: UserRegistrations → Partition key: token (String)
3. Table settings: Customize settings → Capacity mode: On-demand
4. Click Create table → wait for status: Active

Enable TTL

1. Click the UserRegistrations table → Additional settings tab
2. Time to Live (TTL) → Enable → TTL attribute name: ttl → Enable TTL

■ **NOTE:** TTL uses a Unix timestamp. Lambda sets this to 24 hours from submission time. Records never confirmed are automatically deleted by DynamoDB.

2 AWS IAM
Create the Lambda Execution Role

1. IAM → Roles → Create role → AWS service → Lambda → Next
2. Attach: AmazonSESFullAccess
3. Attach: AmazonDynamoDBFullAccess
4. Attach: AWSLambdaBasicExecutionRole
5. Role name: LambdaRegistrationRole → Create role

3 AWS Lambda
Create RegistrationHandler

This function creates the token, saves PENDING record, and sends the verification email to the user.

1. Lambda → Create function → Author from scratch
2. Function name: RegistrationHandler → Runtime: Python 3.12
3. Execution role: LambdaRegistrationRole → Create function
4. Replace all code with the following (replace email and API URL after Step 5):

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

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

SENDER_EMAIL = 'your-verified-email@example.com'  # replace
API_BASE_URL = 'https://YOUR-API-ID.execute-api.us-east-1.amazonaws.com/prod'  # replace after Step 5

def lambda_handler(event, context):
    headers = {'Access-Control-Allow-Origin': '*'}
    try:
        body  = json.loads(event.get('body', '{}'))
        name  = body.get('name', '').strip()
        email = body.get('email', '').strip()
        if not name or not email:
            return {'statusCode': 400, 'headers': headers,
                    'body': json.dumps({'error': 'Name and email required.'})}
        token    = str(uuid.uuid4())
        ts       = datetime.now(timezone.utc).isoformat()
        ttl_time = int((datetime.now(timezone.utc) + timedelta(hours=24)).timestamp())
        confirm_url = f'{API_BASE_URL}/confirm?token={token}'
        table.put_item(Item={
```

```
            'token': token, 'name': name, 'email': email,
            'status': 'PENDING', 'timestamp': ts, 'ttl': ttl_time
        })
        ses.send_email(
            Source=SENDER_EMAIL,
            Destination={'ToAddresses': [email]},
            Message={
                'Subject': {'Data': 'Please verify your email address'},
                'Body': {'Html': {'Data': f'<h2>Hi {name},</h2>'
                    f'<p>Click below to verify your email:</p>'
                    f'<a href="{confirm_url}" style="background:#1B2A4A;color:#fff;padding:12px 24px;text-decoration:none;">Verify M
                    f'<p style="color:#888">Link expires in 24 hours.</p>'}}
            }
        )
        return {'statusCode': 200, 'headers': headers,
                'body': json.dumps({'message': f'Verification email sent to {email}.'})}
    except Exception as e:
        print(f'Error: {str(e)}')
        return {'statusCode': 500, 'headers': headers,
                'body': json.dumps({'error': str(e)})}
```

5. Click Deploy

■ **WARNING:** Leave API_BASE_URL as a placeholder. Update it after Step 5 when you have the real URL.

4 AWS Lambda
Create ConfirmationHandler

This function runs when the user clicks the verification link. It confirms the token, updates DynamoDB, and emails the owner.

1. Lambda → Create function → ConfirmationHandler → Python 3.12 → LambdaRegistrationRole

2. Replace all code with the following:

```
import json, boto3
import urllib.parse
from datetime import datetime, timezone
from boto3.dynamodb.conditions import Attr

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

SENDER_EMAIL = 'your-verified-email@example.com'  # replace
OWNER_EMAIL  = 'your-verified-email@example.com'  # replace
SUCCESS_URL  = 'http://YOUR-BUCKET-URL/confirmed.html'  # replace after Step 7

def lambda_handler(event, context):
    params = event.get('queryStringParameters') or {}
    token  = params.get('token', '').strip()
    if not token:
        return redirect('Missing token. Use the link from your email.')
    try:
        item = table.get_item(Key={'token': token}).get('Item')
        if not item:
            return redirect('Link is invalid or expired (links expire after 24 hours).')
        if item.get('status') == 'CONFIRMED':
            return redirect(f"Already verified: {item.get('email')}")
        name  = item.get('name', 'User')
        email = item.get('email', '')
        ts    = datetime.now(timezone.utc).isoformat()
        table.update_item(
            Key={'token': token},
            UpdateExpression='SET #s = :c, confirmedAt = :ts REMOVE #ttl',
            ExpressionAttributeNames={'#s': 'status', '#ttl': 'ttl'},
            ExpressionAttributeValues={':c': 'CONFIRMED', ':ts': ts},
```

```
            ConditionExpression=Attr('status').eq('PENDING')
        )
        ses.send_email(
            Source=SENDER_EMAIL,
            Destination={'ToAddresses': [OWNER_EMAIL]},
            Message={
                'Subject': {'Data': f'New verified registration: {name}'},
                'Body': {'Text': {'Data':
                    f'Name: {name}\nEmail: {email}\nConfirmed at: {ts}'
                }}
            }
        )
        return {'statusCode': 302,
                'headers': {'Location': SUCCESS_URL}, 'body': ''}
    except Exception as e:
        print(f'Error: {str(e)}')
        return redirect(str(e))

def redirect(msg):
    return {'statusCode': 302,
            'headers': {'Location': f'{SUCCESS_URL}?msg={urllib.parse.quote(msg)}'},
            'body': ''}
```

3. Click Deploy

5 Amazon API Gateway
Create the API with Two Routes

Create the API

1. API Gateway → Create API → HTTP API → Build
2. Add integration → Lambda → RegistrationHandler → API name: RegistrationAPI → Next
3. Method: POST → Resource path: /register → Next → Stage: prod → Create
4. Copy the Invoke URL from the summary page

Add the GET /confirm route

1. Left sidebar → Routes → Create → Method: GET → path: /confirm → Create
2. Click GET /confirm → Attach integration → Create and attach an integration
3. Integration type: Lambda → select ConfirmationHandler → Create

Enable CORS

1. Left sidebar → CORS → Allow-Origin: * → Allow-Methods: POST, GET, OPTIONS → Save

Update RegistrationHandler

1. Lambda → RegistrationHandler → replace API_BASE_URL with your actual Invoke URL → Deploy

6 Amazon S3
Upload register.html

Create register.html with your registration form. Replace YOUR_API_URL with your Invoke URL from Step 5.

```
<!DOCTYPE html><html><head><meta charset='UTF-8'/>
<title>Register</title></head><body>
<h1>Register</h1>
<label>Name</label>
<input type='text' id='name' placeholder='Your full name'/>
<label>Email</label>
<input type='email' id='email' placeholder='you@example.com'/>
<button onclick='submitForm()'>Send Verification Email</button>
```

```
<p id='status'></p>
<script>
  const API = 'YOUR_API_URL/register'; // replace
  async function submitForm() {
    const name  = document.getElementById('name').value.trim();
    const email = document.getElementById('email').value.trim();
    const s = document.getElementById('status');
    if (!name || !email) { s.textContent = 'Fill in all fields.'; return; }
    s.textContent = 'Sending...';
    const r = await fetch(API, { method: 'POST',
      headers: {'Content-Type':'application/json'},
      body: JSON.stringify({name, email}) });
    const d = await r.json();
    s.textContent = d.message || d.error;
  }
</script></body></html>
```

1. Replace YOUR_API_URL with your Invoke URL
2. Upload register.html to your S3 bucket

7 Amazon S3 Upload confirmed.html

Create confirmed.html — users land here after clicking the verification link.

```
<!DOCTYPE html><html><head><meta charset='UTF-8'/>
<title>Email Verified</title></head><body style='text-align:center;margin-top:80px;'>
<div id='content'>Loading...</div>
<script>
  const msg = new URLSearchParams(window.location.search).get('msg');
  document.getElementById('content').innerHTML = msg
    ? '<h1>Notice</h1><p>' + decodeURIComponent(msg) + '</p>'
    : '<h1>&#x2714; Email Verified!</h1>'
    + '<p>Registration complete. The site owner has been notified.</p>';
</script></body></html>
```

1. Upload confirmed.html to your S3 bucket
2. Lambda → ConfirmationHandler → update SUCCESS_URL to your S3 URL + /confirmed.html → Deploy

8 End-to-End Testing Test the Complete Registration Flow

Happy path test

1. Open your S3 URL + /register.html
2. Enter your name and your verified SES email address → click Send Verification Email
3. Check DynamoDB → UserRegistrations → Explore items — a PENDING record should appear
4. Check your inbox — verification email should arrive within seconds
5. Click Verify My Email in the email
6. Browser should redirect to confirmed.html showing the success message
7. Check your owner inbox — notification email should arrive
8. Check DynamoDB — record should now show status: CONFIRMED and ttl attribute removed

Error path test

1. Click the verification link from the email a second time
2. You should see the already-verified notice — confirms idempotency works

Verification Checklist

- ■ DynamoDB table UserRegistrations created with On-demand capacity
- ■ TTL enabled on the table using attribute name: ttl
- ■ IAM role LambdaRegistrationRole created with SES, DynamoDB, CloudWatch permissions
- ■ RegistrationHandler deployed with correct SENDER_EMAIL and API_BASE_URL
- ■ ConfirmationHandler deployed with correct SENDER_EMAIL, OWNER_EMAIL, SUCCESS_URL
- ■ API Gateway RegistrationAPI has POST /register and GET /confirm routes
- ■ Both routes are integrated with the correct Lambda functions
- ■ CORS enabled for POST, GET, OPTIONS
- ■ register.html and confirmed.html uploaded to S3
- ■ Form shows success message after submit
- ■ Verification email arrives with working link
- ■ DynamoDB shows PENDING record immediately after form submit
- ■ Clicking the link redirects to confirmed.html with success message
- ■ DynamoDB record updated to CONFIRMED and ttl attribute removed
- ■ Owner notification email arrives after verification
- ■ Clicking the link a second time shows already-verified message

---

What You Learned

- • Token-based email verification — the industry-standard pattern for confirming user identity
- • DynamoDB TTL — automatic expiration and deletion of records without scheduled cleanup jobs
- • Conditional DynamoDB writes — ConditionExpression prevents replay attacks and race conditions
- • Two Lambda functions — separation of concerns: register and confirm are distinct responsibilities
- • Two API routes — POST /register starts the flow, GET /confirm completes it from the email link
- • HTML email with SES — sending a branded verification email with a styled button
- • 302 Redirect from Lambda — how to redirect a browser to another page after confirmation
- • URL query string parameters — passing token via ?token=UUID and reading in Lambda
- • Idempotent API design — confirming twice is safe and returns a friendly message
- • Full serverless registration pipeline — production-grade pattern with zero servers and auto-scaling

---

Lab Cleanup

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

#	Resource	How to Delete
1	DynamoDB Table	DynamoDB → Tables → UserRegistrations → Delete
2	RegistrationHandler	Lambda → RegistrationHandler → Actions → Delete
3	ConfirmationHandler	Lambda → ConfirmationHandler → Actions → Delete
4	API Gateway	API Gateway → RegistrationAPI → Actions → Delete

5	IAM Role	IAM → Roles → LambdaRegistrationRole → Delete
6	S3 Files	S3 → your bucket → delete register.html and confirmed.html
7	CloudWatch Log Groups	CloudWatch → delete /aws/lambda/RegistrationHandler and ConfirmationHandler