OTP Authentication System – Complete Project Documentation

#Connection names

Lambda function name—otp-auth-handler
dynamo table name---- otp_user app_client----
My web app - thh9r8 api_name---- otp-auth-api

1. Introduction

Connections Used: Streamlit → API Gateway → AWS Lambda → DynamoDB → Amazon SES

This document presents a detailed explanation of the OTP Authentication System built using AWS serverless services and a Streamlit frontend. The system ensures secure, time-bound authentication using One-Time Passwords (OTP) sent via email.

2. Problem Statement

Connections Used: User → Streamlit UI

Traditional password-based authentication systems are vulnerable to brute-force attacks, password leaks, and phishing. This project eliminates password dependency by implementing OTP-based authentication with limited attempts and expiry.

3. Overall System Architecture

Connections Used: Streamlit ↔ API Gateway ↔ Lambda ↔ DynamoDB & SES

The system follows a fully serverless architecture. Streamlit serves as the frontend, API Gateway exposes REST endpoints, AWS Lambda executes OTP logic, DynamoDB stores OTP records, and Amazon SES delivers OTP emails.

4. Streamlit Frontend

Connections Used: User Browser → Streamlit App → API Gateway

The Streamlit UI collects user email and OTP input. It sends POST requests to API Gateway and displays responses received from the backend. It handles both success and error messages gracefully.

5. API Gateway

Connections Used: Streamlit → API Gateway → Lambda

API Gateway acts as the secure entry point for the system. It routes HTTP POST requests to Lambda using proxy integration and handles CORS to allow browser-based access.

6. AWS Lambda Function

Connections Used: API Gateway → Lambda → DynamoDB / SES

The Lambda function is the core of the system. It generates OTPs, validates user input, enforces expiry and attempt limits, stores data in DynamoDB, and sends emails through Amazon SES.

7. DynamoDB Database

Connections Used: Lambda → DynamoDB

DynamoDB stores OTP details with email as the partition key. Attributes include OTP value, expiry timestamp, attempts count, status, and creation time. TTL automatically deletes expired OTP records.

8. DynamoDB TTL (Time To Live)

Connections Used: DynamoDB → Automatic Expiry Engine

TTL is configured on the otp_expiry attribute. Once the expiry time is reached, DynamoDB automatically removes the item without manual cleanup, improving efficiency and security.

9. Amazon SES (Email Service)

Connections Used: Lambda → Amazon SES → User Email Inbox

Amazon SES is used to send OTP emails. In sandbox mode, both sender and receiver emails must be verified. SES ensures reliable email delivery with low latency.

10. IAM Roles & Permissions

Connections Used: Lambda Execution Role → DynamoDB & SES

The Lambda execution role grants permissions to DynamoDB (PutItem, GetItem, UpdateItem) and SES (SendEmail). Least-privilege access is followed for security.

11. OTP Workflow

Connections Used: User → Streamlit → API Gateway → Lambda → DynamoDB → SES

When a user requests an OTP, Lambda generates and stores it, then sends it via SES. During verification, Lambda validates OTP correctness, expiry, and attempt count before confirming authentication.

12. Security Measures

Connections Used: Lambda → DynamoDB TTL → IAM

Security features include OTP expiry, limited verification attempts, server-side validation, IAM access control, and removal of expired records.

13. Error Handling & Logging

Connections Used: Lambda → CloudWatch Logs

All errors are handled gracefully. Lambda logs important execution details to CloudWatch, enabling monitoring and debugging.

14. Testing Strategy

Connections Used: Streamlit UI / Lambda Test Events / DynamoDB Console

Testing includes sending OTPs, verifying correct and incorrect OTPs, checking expiry behavior, attempt limit enforcement, and validating database records.

15. Deployment Process

Connections Used: AWS Console → Lambda → API Gateway → Streamlit

Deployment involves creating DynamoDB tables, configuring SES, deploying Lambda, setting up API Gateway, and running the Streamlit frontend.

16. Limitations

Connections Used: SES Sandbox / Email-only OTP

The system currently supports email-based OTP only and is restricted by SES sandbox limitations until production access is granted.

17. Future Enhancements

Connections Used: SNS / JWT / MFA Integration

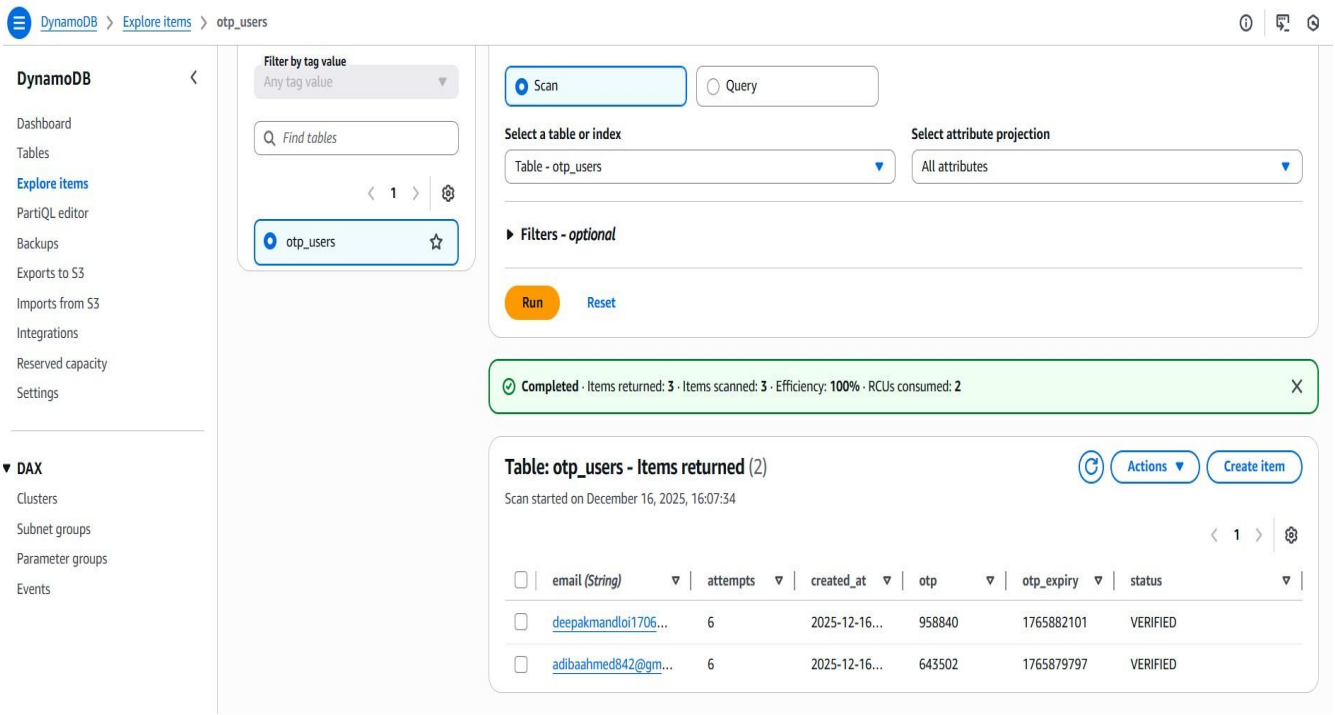
Planned improvements include SMS OTP via SNS, JWT-based session management, multifactor authentication, and admin dashboards.

18. Conclusion

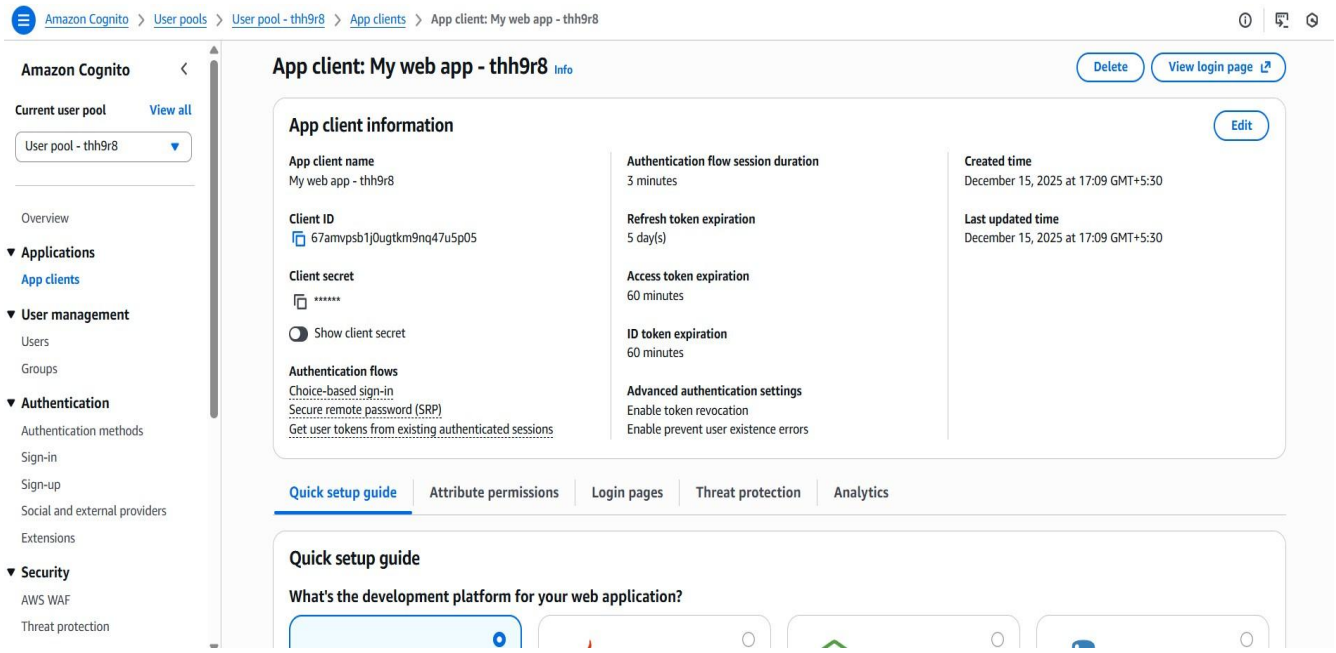
Connections Used: Complete Serverless Authentication System

This project demonstrates a secure, scalable, and cost-effective OTP authentication system using AWS serverless technologies and Streamlit.

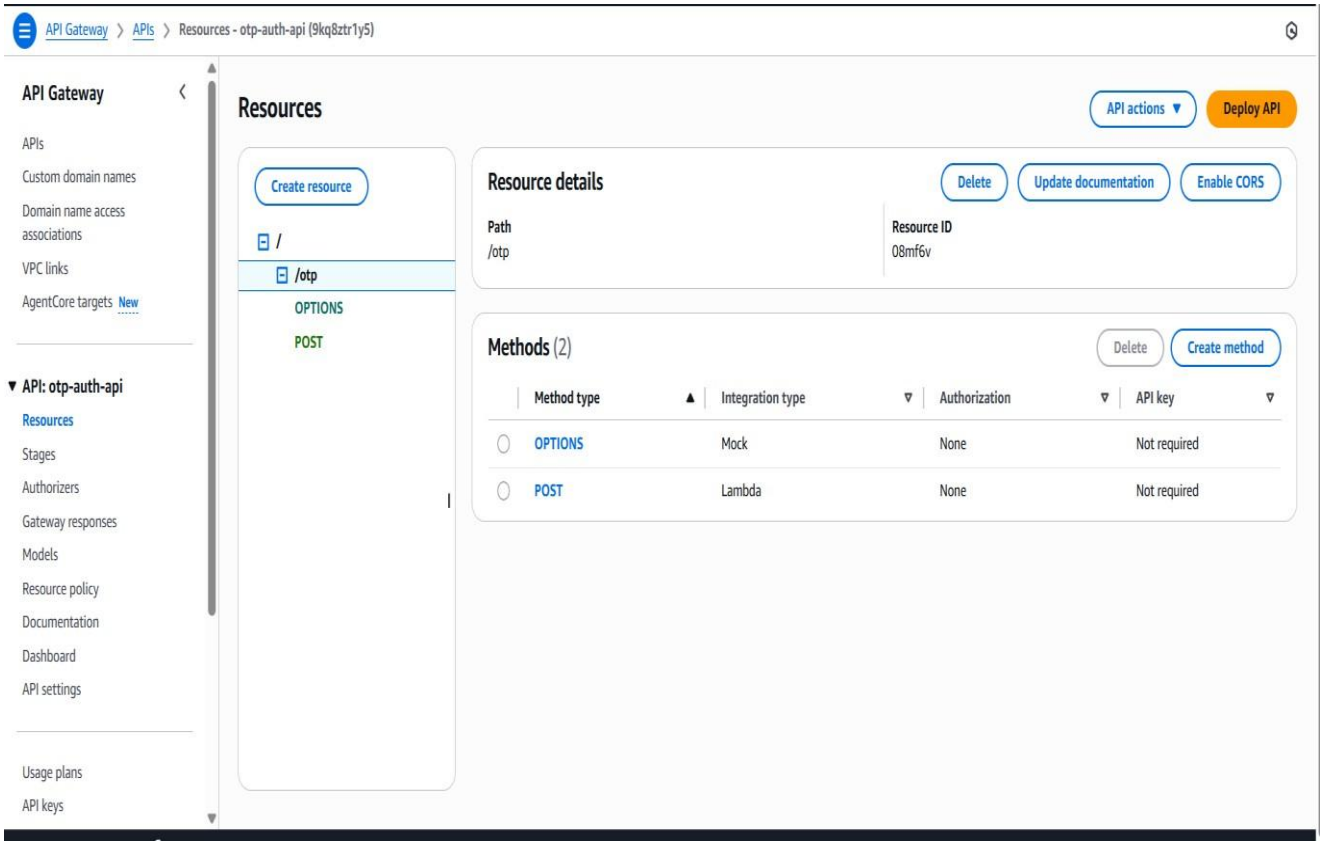
#DYNAMODB--IMG



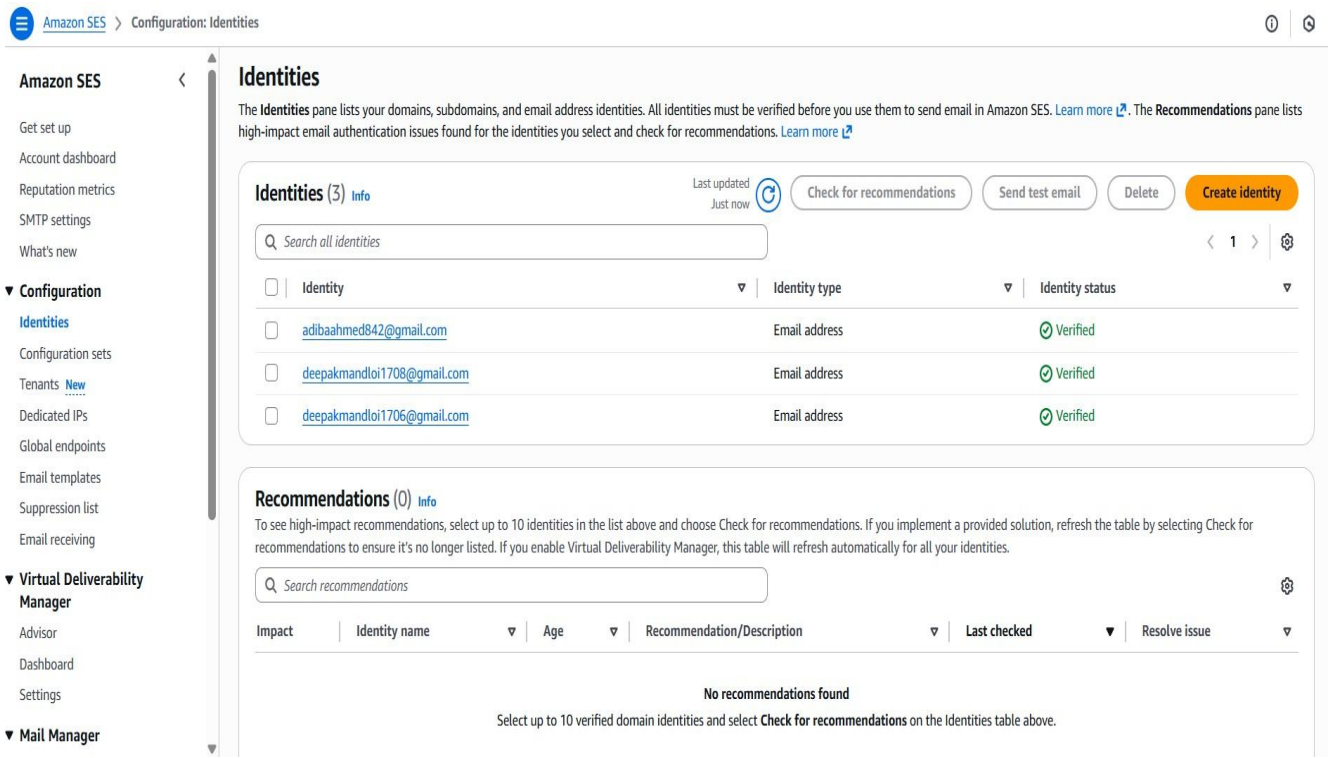
#COGNITO—IMG



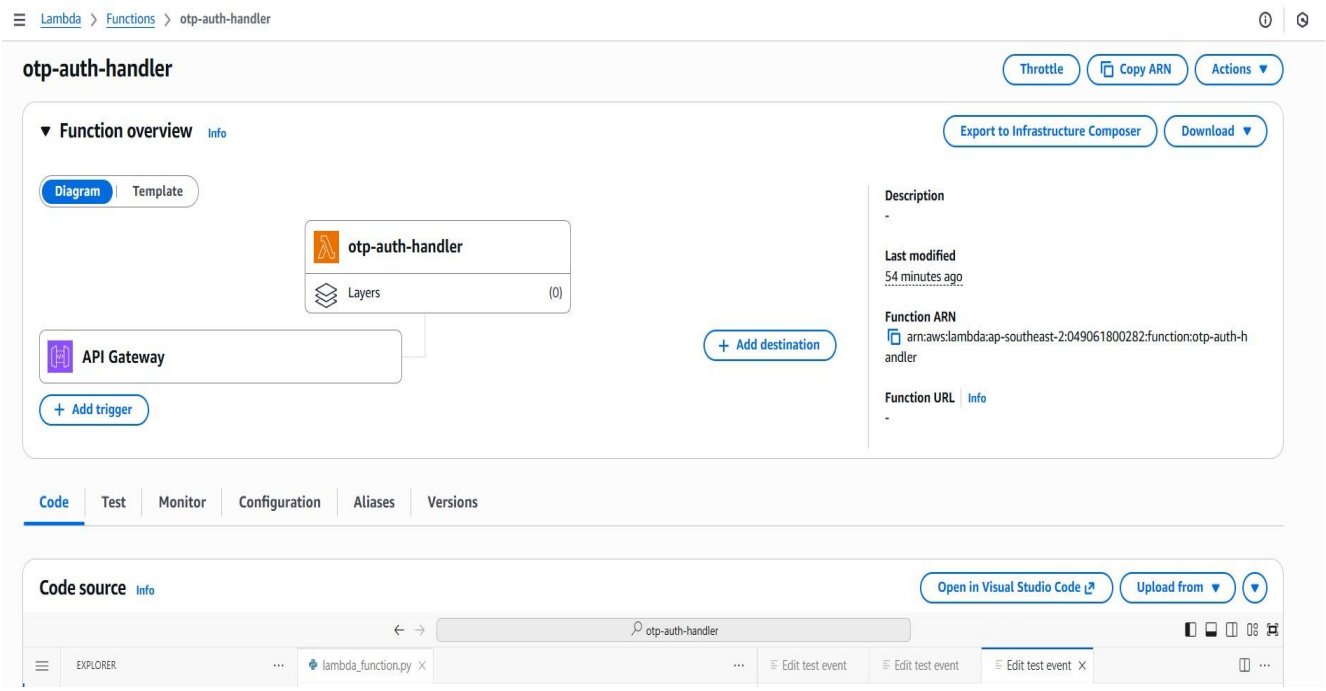
#API_GATEWAY---IMG



#SES--IMG



#LAMBDA---IMG



#APP.py

import streamlit as st

import requests import

json

API_URL = "https://9kq8ztr1y5.execute-api.ap-southeast-2.amazonaws.com/prod/otp"

st.set_page_config(page_title="OTP Authentication", layout="centered") st.title("OTP

Authentication System")

email = st.text_input("Enter your email")

```
def show_response(r):
    try:
        data = r.json()        body =
json.loads(data["body"])        msg =
body.get("message", "No message")     except
Exception:
        msg = r.text or "Invalid server response"

    if r.status_code == 200:
        st.success(msg)
else:
        st.error(msg)
# SEND OTP if st.button("Send OTP",
disabled=not email):
    r = requests.post(API_URL, json={
        "action": "send_otp",
        "email": email
    })
    show_response(r)

st.divider()

otp = st.text_input("Enter OTP")
```

```
# VERIFY OTP if st.button("Verify OTP", disabled=not
(email and otp)):
    r = requests.post(API_URL, json={
        "action": "verify_otp",
        "email": email,
        "otp": otp
    })
    show_response(r)
```

#LAMBDA FUNCTION

```
import json import boto3
import time import random
import re from datetime
import datetime

dynamodb = boto3.resource("dynamodb") table =
dynamodb.Table("otp_users") ses = boto3.client("ses",
region_name="ap-southeast-2")

SENDER_EMAIL = "deepakmandloi1708@gmail.com"
OTP_VALIDITY_SECONDS = 15 * 60
MAX_ATTEMPTS = 6
EMAIL_REGEX = r"[^@]+@[^@]+\.[^@]+"
```

```
def generate_otp():
    return str(random.randint(100000, 999999))

def send_otp_email(email, otp):
ses.send_email(
        Source=SENDER_EMAIL,
        Destination={"ToAddresses": [email]},
        Message={
            "Subject": {"Data": "Your OTP Code"},
             "Body": {"Text": {"Data": f"Your OTP is {otp}. Valid for 15 minutes."}}
        }
    )

def response(code, msg):
    return {
        "statusCode": code,
        "headers": {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*"
        },
        "body": json.dumps({"message": msg})
    }

def lambda_handler(event, context):
```

```
   body = {}     if "body" in event and
event["body"]:
       try:
          body = json.loads(event["body"])
except Exception:
         return response(400, "Invalid JSON body")
  else:
      body = event  # fallback
  action = body.get("action") email
= body.get("email")    otp_input =
body.get("otp")

   if not action or not email:
       return response(400, "Action and email are required")

   if not re.match(EMAIL_REGEX, email):
       return response(400, "Invalid email format")

   if action == "send_otp":        otp = generate_otp()
expiry = int(time.time()) + OTP_VALIDITY_SECONDS

       table.put_item(
          Item={
             "email": email,
```

```
            "otp": otp,
            "otp_expiry": expiry,   # TTL
            "attempts": 0,
            "status": "PENDING",
           "created_at": datetime.utcnow().isoformat()
        }
    )
     send_otp_email(email, otp)     return
  response(200, "OTP sent successfully")

 if action == "verify_otp":

    if not otp_input:
        return response(400, "OTP is required")

    res = table.get_item(Key={"email": email})
user = res.get("Item")

    if not user:
        return response(404, "OTP not found")

    if user["attempts"] >= MAX_ATTEMPTS:
        return response(403, "OTP attempts exceeded (6)")
```

```
        if int(time.time()) > user["otp_expiry"]:
return response(403, "OTP expired")

        if otp_input != user["otp"]:
 table.update_item(
              Key={"email": email},
              UpdateExpression="SET attempts = attempts + :inc",
              ExpressionAttributeValues={":inc": 1}
            )
            return response(401, "Invalid OTP")

       table.update_item(
            Key={"email": email},
            UpdateExpression="SET #s = :v",
            ExpressionAttributeNames={"#s": "status"},
            ExpressionAttributeValues={":v": "VERIFIED"}
        )

        return response(200, "OTP verified successfully")

    return response(400, "Invalid action")

-------------------------------------------------------------------------------------------

#JSON TEST EVENTS
```

test3--- {

"action": "send_otp",

"email": "deepakmandloi1708@gmail.com"

}

Test4----

{

"action": "send_otp",

"email": "deepakmandloi1706@gmail.com"

}

Test5----

{

"body":
"{\"action\":\"send_otp\",\"email\":\"deeoakmandloi1708@gmail.com\"}"

}