# CLOUD-BASED USER REGISTRATION AND NOTIFICATION SYSTEM

## Project Overview

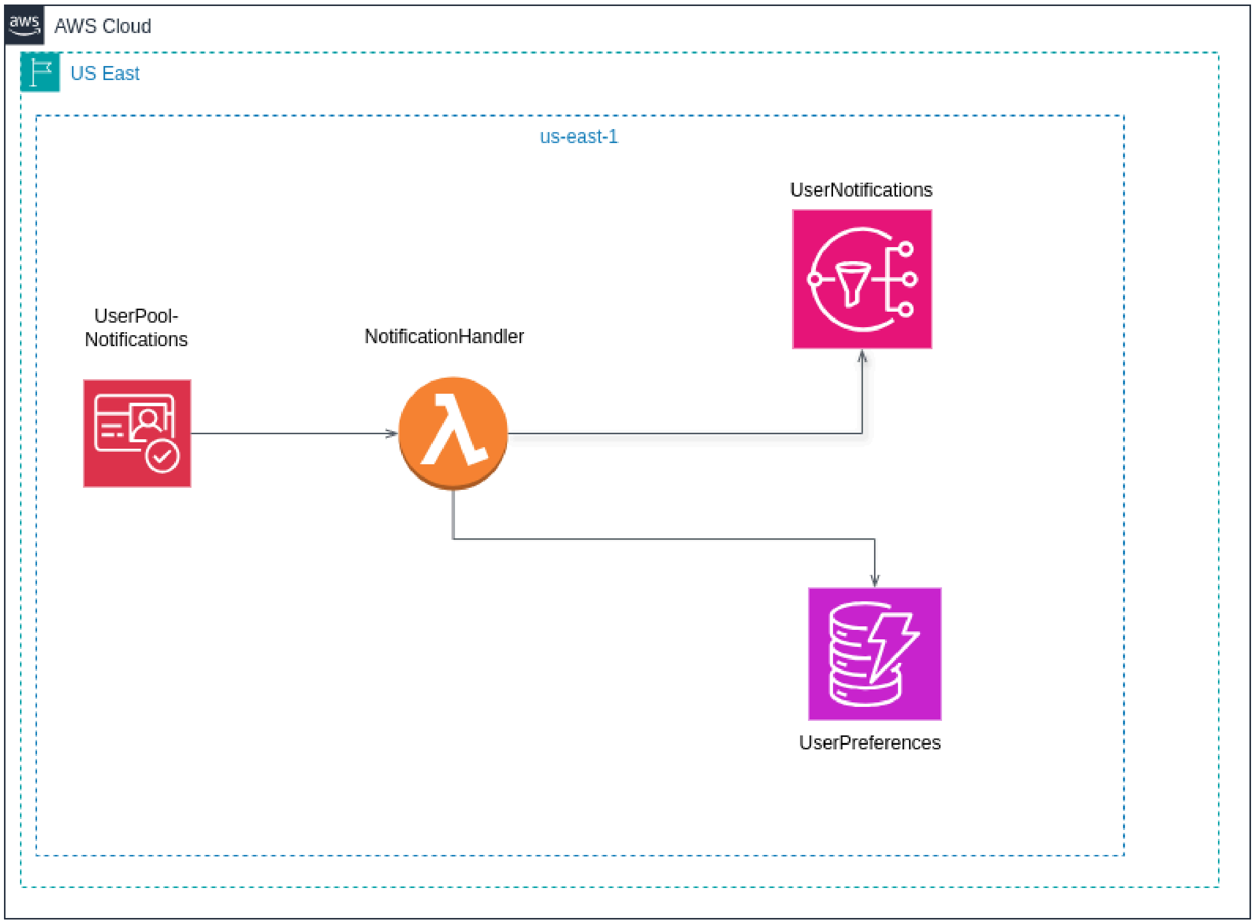
This project implements a user registration and notification system using AWS services. It automates the user sign-Up process, confirms user email address, stores user data in DynamoDB and sends notification emails via Amazon SNS. This system is designed to provide a seamless user experience while ensuring secure and scalable data management.

## Architecture

The architecture of the system consists of the following components:

- [ ] **AWS Cognito:** Manages user authentication and provides a user pool for registration
- [ ] **AWS Lambda:** Executes backend logic to handle post-confirmation events for user registration.
- [ ] **DynamoDB:** A NoSQL database for storing user details and preferences.
- [ ] **Amazon SNS:** A messaging service used to send email notifications to users.

## Architecture Diagram:



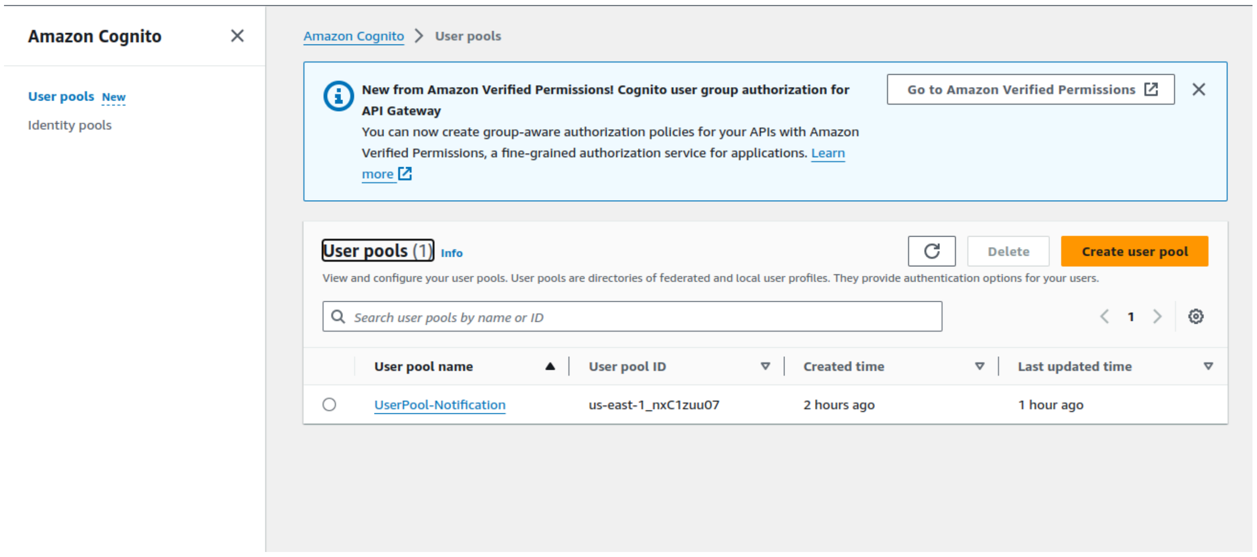
# Setup Instructions

## Prerequisites

- An AWS account
- AWS CLI installed and configured
- Python installed for Lambda functions

## Step 1: Create a Cognito User Pool

1. Navigate to the AWS Cognito console.
2. Click on “Create a user Pool”
3. Configure the pool settings:
   - Require email and username as a sign-in attribute
   - Enable email verification.
4. Create the pool and note the User Pool ID.


Amazon Cognito
User pools New
Identity pools
Amazon Cognito > User pools
Go to Amazon Verified Permissions
Delete
Create user pool
Info
Search user pools by name or ID

New from Amazon Verified Permissions! Cognito user group authorization for API Gateway

You can now create group-aware authorization policies for your APIs with Amazon Verified Permissions, a fine-grained authorization service for applications. Learn more

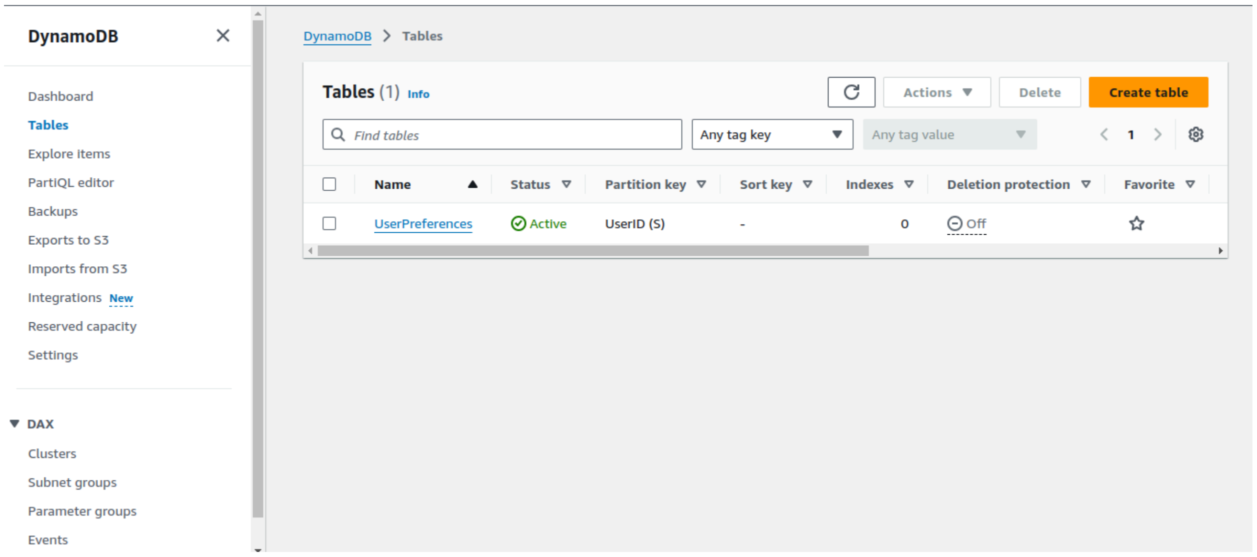
User pools (1)

View and configure your user pools. User pools are directories of federated and local user profiles. They provide authentication options for your users.

| User pool name | User pool ID | Created time | Last updated time |
|---|---|---|---|
| UserPool-Notification | us-east-1_nxC1zuu07 | 2 hours ago | 1 hour ago |

## Step 2: Create a DynamoDB Table

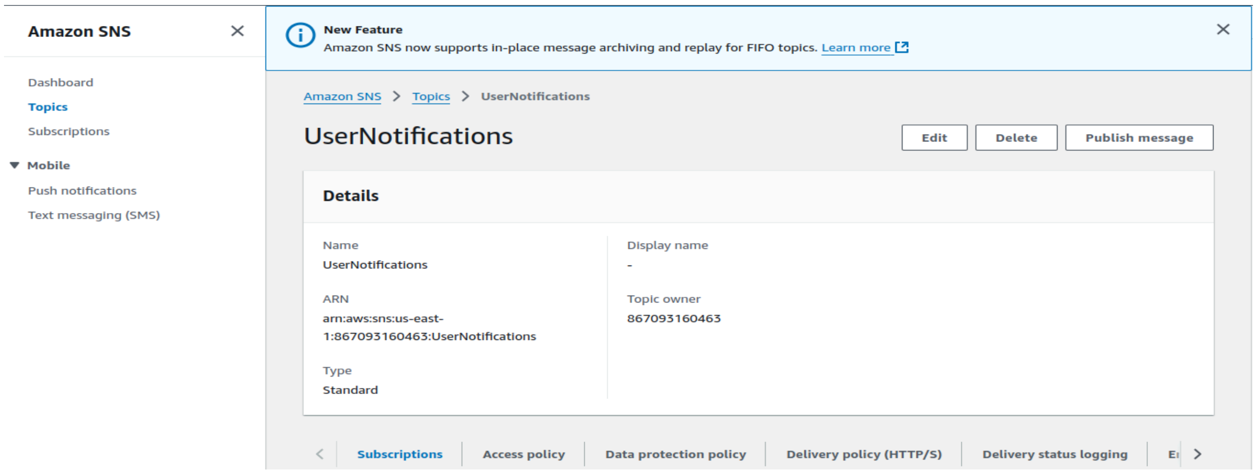
1. Navigate to the DynamoDB console.
2. Click on “Create table”.
3. Set the table name . Ex. UserPreference.
4. Set the primary key. Ex. UserID
5. Create the table.



| | Name | Status | Partition key | Sort key | Indexes | Deletion protection | Favorite |
|---|---|---|---|---|---|---|---|
| ☐ | UserPreferences | Active | UserID (S) | - | 0 | Off | ☆ |

## Step 3: Create an SNS Topic

1. Navigate to the SNS console.
2. Click on “Create topic.”
3. Select “Standard” and name your topic. Ex. UserNotifications.
4. Note the Topic ARNfor later use.



New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

UserNotifications

Details

Name
UserNotifications

ARN
arn:aws:sns:us-east-1:867093160463:UserNotifications

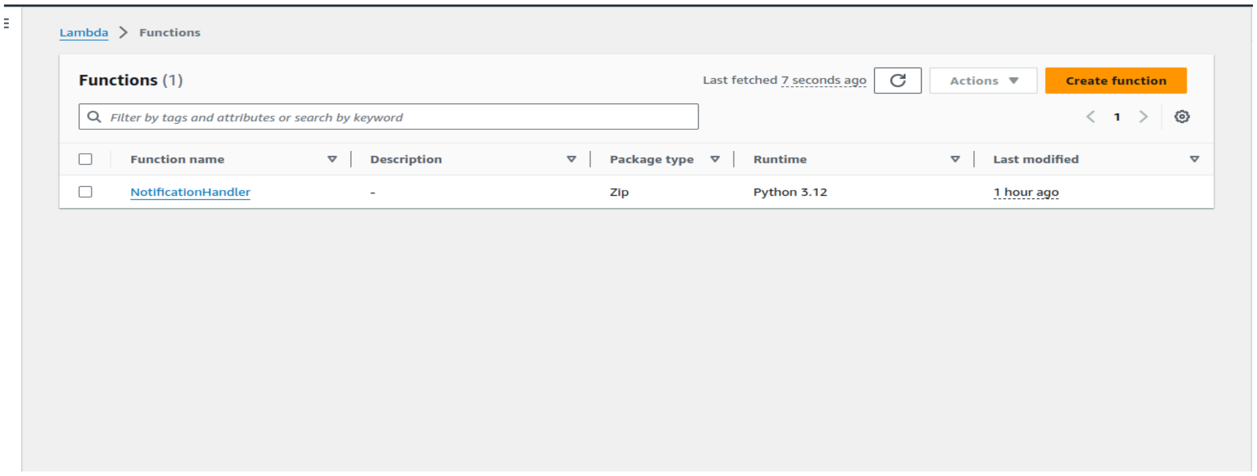
Type
Standard

Display name
-

Topic owner
867093160463

## Step 4: create a Lambda Function

1. Navigate to the Lambda console.
2. Click on “Create function.
3. Choose “Author from scratch”
4. Name your function. Ex. NotificationHandler
5. set the runtime to Python 3.x.
6. Set up the execution role with permissions for DynamoDB and SNS.
7. Copy and paste the provided Lambda Code into the function code editor.
8. Configure the trigger to the Cognito User pool post-confirmation event.


Lambda > Functions
Last fetched 7 seconds ago
Actions
Create function
Filter by tags and attributes or search by keyword
1

Functions (1)

| Function name | Description | Package type | Runtime | Last modified |
|---|---|---|---|---|
| NotificationHandler | - | Zip | Python 3.12 | 1 hour ago |

## Step 5: Set Permissions

Ensure the lambda function has permissions to access the DynamoDB table and publish to the SNS topic by attaching the following policies to its execution role:

- LambdaDynamoDBWritePolicy.json
- AmazonSNSFullAccess

LambdaDynamoDBWritePolicy.json

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Sid": "Statement1",
                        "Effect": "Allow",
                        "Action": "dynamodb:PutItem",
                        "Resource":
"arn:aws:dynamodb:us-east-1:867093160463:table/UserPreferences"
                }
        ]
}
```

## Functionality

- **User Registration:** Users can register using their email and username.
- **Email Verification:** Upon registration, a verification code is sent to  the user's email.
- **Data Storage:** once the user confirms their email, their details are stored in DynamoDB.
- **Notification:** A welcome message is sent to the user's email via SNS.

## AWS Services Used

- **AWS Cognito:**User authentication and management
- **AWS Lambda:**Serverless function to process user confirmation and store data
- **DynamoDB:** Data storage for user preferences.
- **Amazon SNS:** Notification service for sending email

## Code Explanation

The lambda function processes the post-confirmation event from Cognito and performs the following actions:

- Extract User Details: Retrieves the username and email from the event object

- Store User in DynamoDB: Saves the user details in the UserPreferences table.
- Send Notification: Publishes a welcome message to the SNS topic.

Code Snippet:

```
import json
import boto3
import logging

# Initialize logging
logging.basicConfig(level=logging.INFO)

# Initialize DynamoDB and SNS clients
dynamodb = boto3.resource('dynamodb')
sns = boto3.client('sns')

def lambda_handler(event, context):
    logging.info("Received event: %s", json.dumps(event))

    try:
        user_id = event['userName']
        email = event['request']['userAttributes']['email']

        logging.info(f"User confirmed: {user_id}, Email: {email}")

        # Access DynamoDB Table
        table = dynamodb.Table('UserPreferences')

        # Store user details in DynamoDB
        table.put_item(
            Item={
                'UserID': user_id,
                'Email': email,
                'NotificationPreference': 'Email'
            }
        )
        logging.info("User details saved to DynamoDB")
        # Subscribe the user's email to the SNS topic
        try:
            sns.subscribe(

TopicArn='arn:aws:sns:us-east-1:867093160463:UserNotifications',  #
Replace with your SNS Topic ARN
                Protocol='email',
                Endpoint=email
            )
            logging.info(f"User {email} subscribed to SNS topic")
        except Exception as sub_error:
            logging.error(f"Error subscribing user {email} to SNS
topic: {str(sub_error)}")
            # You can choose to continue or return here if you don't
want the process to continue.
            # return event

        # Send notification via SNS
        message = f"Hello {user_id}, your account has been
successfully created!"
        response = sns.publish(

TopicArn='arn:aws:sns:us-east-1:867093160463:UserNotifications',  #
Replace with your SNS Topic ARN
            Message=message,
            Subject="Account Created"
        )
        logging.info("SNS publish response: %s", response)

        # Return the event as required by Cognito Post Confirmation
trigger
        return event

    except Exception as e:
        logging.error("Error occurred: %s", str(e))
        return {
            'statusCode': 500,
            'body': json.dumps(f"An error occurred: {str(e)}")
        }
```

## User Subscription Automation

To automate user email subscription to the SNS topic, you can modify the Lambda function to include the subscription process. After storing the user details, add the following code:

```
# Subscribe the user's email to the SNS topic
sns.subscribe(
    TopicArn='arn:aws:sns:us-east-1:867093160463:UserNotifications',
    Protocol='email',
    Endpoint=email
)
```

## Testing

To test the system :

- Use the AWS Cognito console to create a new user.
- Check that the confirmation email is sent.
- After confirming the email, verify that the user data appears in the DynamoDB table and the welcome message is sent via SNS.

## Troubleshooting

- **Log:** Check AWS Cloudwatch logs for lambda to diagnose issues.
- **Permissions:** Ensure the lambda function has the necessary permissions for DynamoDB and SNS.
- **SNS Subscription:**Verify that the user’s email is subscribed to the SNS topic.

## Future Improvements

- **User Preferences:**Extends the system to allow users to set notification preferences(e.g, SMS, push notifications)
- **Error Handling:** Improve error handling and logging for better debugging.

- **Analytics:** Implement analytics to track user engagement with notifications.

## Conclusion

This project provides a robust solution for user registration and notification management using AWS services.By automating key processes, it enhances user experience while ensuring secure and efficient data management.

Here are some real-life applications of a user registration and notification system like the one above:

1. **E-Commerce Platforms**:
    - User registration allows customers to create accounts for a personalized shopping experience. Email notifications can inform them about order confirmations, shipping updates, and promotional offers.
2. **Social Media Applications**:
    - Users can sign up to create profiles, connect with others, and share content. Email confirmations help verify accounts, while notifications keep users informed about messages, likes, or comments.
3. **Online Learning Platforms**:
    - Students can register for courses, and upon enrollment, receive confirmation emails. Notifications about course updates, deadlines, or new content keep learners engaged.
4. **Health and Fitness Apps**:
    - Users register to track their health goals and receive tailored notifications about workouts, nutrition tips, or appointment reminders for virtual consultations with trainers or doctors.
5. **Event Management Systems**:
    - Attendees can sign up for events, receiving email confirmations and reminders. Notifications can provide updates on event schedules, speakers, or any changes to the event.
6. **News and Content Subscription Services**:
    - Users can register to subscribe to newsletters or notifications about new articles, updates, or special offers, enhancing user engagement and content consumption.
7. **Gaming Platforms**:
    - Players can create accounts to save their progress. Notifications can inform them about game updates, promotions, or in-game events, fostering community engagement.
8. **Job Portals**:
    - Users can sign up for job alerts based on their preferences. Confirmation emails verify their registrations, while notifications keep them updated on new job postings that match their criteria.
9. **Customer Support Services**:

- Users register to access support services, receiving confirmation emails and updates about ticket status or resolutions. Notifications can guide users through troubleshooting processes.

10. **Financial Services**:
    - Users can create accounts to manage their finances, receive confirmation emails for transactions, and notifications for account activity, alerts, or payment reminders.