**Project: TaskFlow Pro - Serverless Task Management System**Every place that got your-student-id, place inside it the current date in this format (As example: 20251806

**Architecture Overview:**

A serverless task management system where:

* **Regular Users**: Create/manage their own tasks, receive notifications
* **Admins**: View all tasks, access analytics dashboard, manage system settings

**AWS Services (11 services):**

1. **S3** - Static website hosting
2. **Cognito** - User authentication with groups
3. **Lambda** - Backend processing
4. **DynamoDB** - Task storage
5. **API Gateway** - REST API
6. **CloudWatch** - Logs & monitoring
7. **SNS** - Email notifications
8. **SQS** - Task processing queue
9. **EventBridge** - Scheduled reports
10. **Systems Manager Parameter Store** - Configuration
11. **CloudTrail** - Audit logging

**Foundation Setup**

**Step 1: Create S3 Buckets**

**1.1 Create Website Bucket:**

1. Go to S3 Console
2. Click "Create bucket"
3. Bucket name: taskflow-website-[your-student-id]
4. Uncheck "Block all public access"
5. Acknowledge the warning
6. Click "Create bucket"

**1.2 Enable Static Website Hosting:**

1. Click on your bucket
2. Go to "Properties" tab
3. Scroll to "Static website hosting"
4. Click "Edit"
5. Enable: Static website hosting
6. Hosting type: Host a static website
7. Index document: index.html
8. Error document: error.html
9. Save changes

**1.3 Create Bucket Policy:**

1. Go to "Permissions" tab
2. Edit Bucket policy
3. Paste this policy (replace YOUR-BUCKET-NAME):

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::YOUR-BUCKET-NAME/\*"

}

]

}

**Step 2: Create Website Files:**Already created: error.html**,** index.html**,** css/style.css  
Create: js/config.js **config.js:**// Configuration will be updated after Cognito setup

const config = {

cognito: {

domain: '', // Will be filled after Cognito setup

clientId: '', // Will be filled after Cognito setup

redirectUri: '', // Will be S3 website URL

region: 'us-east-1'

},

api: {

endpoint: '' // Will be filled after API Gateway setup

}

};

**Step 3: Upload Files to S3**

1. Go to your website bucket
2. Click "Upload"
3. Create folder structure:
   * Create folder "css"
   * Create folder "js"
4. Upload files:
   * index.html to root
   * error.html to root
   * style.css to css folder
   * config.js to js folder

**Step 4: Setup Cognito User Pool**

1. Go to Cognito Console
2. User Pool
3. Create User Pool
   1. Choose SPA
   2. Name: TaskFlowWebApp
   3. Sign-in: email
   4. Sign-up: email, name
   5. Return Url put the link from S3 to index.html
   6. Create
4. Click at User Pool that was just created
   1. Click Rename
   2. Rename to: TaskFlowUserPool
5. Click at TaskFlowWebApp
   1. Login Pages
   2. At Managed Login Pages Configuration Click Edit
   3. For Sign-out URL put the link from S3 to index.html
   4. At OpenId Connect Scopes Choose: openid, email, profile
6. Click at Groups
   1. For Every Group choose LabRole
   2. Create
      1. Name: users, Description: “Regular users”
   3. Create
      1. Name: admins, Description: “Administrator users”

**Step 5: Create Lambda for Auto-assign Group**

**5.1 Create Lambda Function:**

1. Go to Lambda Console
2. Click "Create function"
3. Function name: TaskFlow-AutoAssignGroup
4. Runtime: Python 3.9
5. Architecture: x86\_64
6. Permissions: Use existing role - LabRole
7. Click "Create function"

**5.2 Add Function Code:**

import json

def lambda\_handler(event, context):

# Auto-assign new users to 'users' group

return event

**5.3 Add Post Confirmation Trigger:**

1. Go back to Cognito User Pool
2. Go to Extensions
3. Click "Add Lambda trigger"
4. Trigger type: Post confirmation
5. Lambda function: TaskFlow-AutoAssignGroup
6. Click "Add Lambda trigger"

**Step 6: Update Lambda to Assign Group**

Go back to Lambda and update the code (The code is provided in a file named: TaskFlow-AutoAssignGroup or just use this:) :

import json

import boto3

def lambda\_handler(event, context):

# Get the user pool ID and username from the event

user\_pool\_id = event['userPoolId']

username = event['userName']

# Create Cognito client

cognito = boto3.client('cognito-idp')

try:

# Add user to 'users' group

cognito.admin\_add\_user\_to\_group(

UserPoolId=user\_pool\_id,

Username=username,

GroupName='users'

)

print(f"Successfully added {username} to users group")

except Exception as e:

print(f"Error adding user to group: {str(e)}")

# Don't fail the trigger

# Return the event to continue

return event

**Step 7: Create Lambda for Email Validation**

**7.1 Create Lambda Function:**

1. Go to Lambda Console
2. Click "Create function"
3. Function name: TaskFlow-PreSignupValidation
4. Runtime: Python 3.9
5. Architecture: x86\_64
6. Permissions: Use existing role - LabRole
7. Click "Create function"

**7.2 Add Function Code (Its found in a file named TaskFlow-AutoAssignGroup.py):**

import json

import boto3

def lambda\_handler(event, context):

# Get the user pool ID and username from the event

user\_pool\_id = event['userPoolId']

username = event['userName']

# Create Cognito client

cognito = boto3.client('cognito-idp')

try:

# Add user to 'users' group

cognito.admin\_add\_user\_to\_group(

UserPoolId=user\_pool\_id,

Username=username,

GroupName='users'

)

print(f"Successfully added {username} to users group")

except Exception as e:

print(f"Error adding user to group: {str(e)}")

# Don't fail the trigger

# Return the event to continue

return event

**7.3 Add Pre sign-up Trigger:**

1. Go back to Cognito User Pool
2. Go to Extensions
3. Click "Add Lambda trigger"
4. Trigger type: Pre sign-up
5. Lambda function: TaskFlow-PreSignupValidation
6. Click "Add Lambda trigger"

**Step 8: Create DynamoDB Tables**

**8.1 Tasks Table:**

1. Go to DynamoDB Console
2. Click "Create table"
3. Table name: TaskFlow-Tasks
4. Partition key: userId (String)
5. Sort key: taskId (String)
6. Settings: Default settings
7. Click "Create table"

**8.2 Analytics Table:**

1. Create another table
2. Table name: TaskFlow-Analytics
3. Partition key: date (String)
4. Sort key: metric (String)
5. Click "Create table"

**Step 9: Update config.js**

1. Get your S3 website URL from S3 bucket properties
2. Update js/config.js:

const config = {

cognito: {

domain: 'domain.auth.us-east-1.amazoncognito.com',

clientId: '[YOUR-CLIENT-ID]',

redirectUri: 'https://[YOUR-S3-WEBSITE-URL]/index.html',

region: 'us-east-1'

},

api: {

endpoint: ''

}

};

1. Re-upload config.js to S3

(For the domain provide the Url without https://), (It comes by default with https://)  
(The cognito domain is the default one, not a custom domain)

**Step 10: Create Test Users**

**10.1 Create Admin User:**

1. In Cognito User Pool, go to "Users" tab
2. Click "Create user"
3. User information:
   * Don't send an invitation (The default one)
   * Email: [admin@gmail.com](mailto:admin@gmail.com)
   * Set password as: 123456aA!
4. Click "Create user"
5. Click at the User -> edit -> Update the name to be the same as the email
6. Go to user details or Click at the User, Groups tab
7. Add to group: admins

**10.2 Create Regular User:**

1. Create another user:
   * Don't send an invitation (The default one)
   * Email: [user@gmail.com](mailto:user@gmail.com)
   * Set password as: 123456aA!
2. Click "Create user"
3. Click at the User -> edit -> Update the name to be the same as the email
4. Go to user details or Click at the User, Groups tab
5. Add to group: users

**Lambda Functions & API Gateway**

**Step 11: Create Task Handler Lambda**

**11.1 Create Function:**

1. Lambda Console → Create function
2. Function name: TaskFlow-TaskHandler
3. Runtime: Python 3.9
4. Use existing role: LabRole
5. Add Code (Found inside a file called: TaskFlow-TaskHandler.py)

**Step 12: Create Analytics Lambda**

**12.1 Create Function:**

1. Function name: TaskFlow-Analytics
2. Runtime: Python 3.9
3. Use existing role: LabRole
4. Add Code (Found inside a file called: TaskFlow-Analytics.py)

**Step 13: Create API Gateway**

**13.1 Create REST API:**

1. Go to API Gateway Console
2. Click "Create API"
3. Choose REST API (not private)
4. API name: TaskFlowAPI
5. Endpoint Type: Regional
6. Click "Create API"

**13.2 Create Authorizer:**

1. In left menu, click "Authorizers"
2. Click "Create Authorizer"
3. Name: TaskFlowCognitoAuth
4. Type: Cognito
5. Cognito User Pool: Select your TaskFlowUserPool
6. Token Source: Authorization
7. Click "Create"

**13.3 Create Resources:**

1. Click "Resources" in left menu
2. Select "/" root
3. Actions → Create Resource
4. Resource Name: tasks
5. Resource Path: /tasks
6. Enable CORS: Yes
7. Click "Create Resource"

**13.4 Create Methods for /tasks:**

1. Select /tasks
2. Actions → Create Method → GET
   * Integration type: Lambda Function
   * Use Lambda Proxy integration: Yes
   * Lambda Function: TaskFlow-TaskHandler
   * Click "Save", OK to add permission
3. Method Request:
   * Authorization: TaskFlowCognitoAuth
4. Repeat for POST method

**13.5 Create {taskId} resource:**

1. Select /tasks
2. Actions → Create Resource
3. Resource Name: {taskId}
4. Resource Path: {taskId}
5. Enable CORS: Yes

**13.6 Add DELETE method to {taskId}:**

1. Select /{taskId}
2. Actions → Create Method → DELETE
3. Same settings as above

**13.6 Add Update method to {taskId}:**

1. Select /{taskId}
2. Actions → Create Method → PUT
3. Same settings as above

**13.8 Create /admin/analytics:**

1. Create /admin resource
2. Create /analytics under /admin
3. Add GET method with TaskFlow-Analytics Lambda

**13.9 Deploy API:**

1. Actions → Deploy API
2. Deployment stage: New Stage
3. Stage name: prod
4. Click "Deploy"
5. Note the Invoke URL

**Step 14: Update config.js with API Endpoint**

1. Update js/config.js (API endpoint, (Invoke URL)):

const config = {

cognito: {

domain: 'domain.auth.us-east-1.amazoncognito.com',

clientId: '[YOUR-CLIENT-ID]',

redirectUri: 'https://[YOUR-S3-WEBSITE-URL]/index.html',

region: 'us-east-1'

},

api: {

endpoint: 'https://[YOUR-API-ID].execute-api.us-east-1.amazonaws.com/prod'

}

};

**Frontend JavaScript & Additional Services**

**Step 15: Create auth.js**

**js/auth.js (Already created, I provided the file)**

**Step 16: Create app.js**

**js/app.js (Already created, I provided the file)**

**Step 17: Upload Updated Files to S3**

Upload all JavaScript files to the js folder in your S3 bucket.

**Additional Services**

**Step 18: Create SNS Topic**

1. Go to SNS Console
2. Create topic:
   * Type: Standard
   * Name: TaskFlow-Notifications
3. Create subscription:
   * Protocol: Email
   * Endpoint: your email

**Step 19: Create SQS Queue**

1. Go to SQS Console
2. Create queue:
   * Name: TaskFlow-ProcessingQueue
   * Type: Standard

**Step 20: Create Parameter Store Config**

1. Go to Systems Manager → Parameter Store
2. Create parameter:
   * Name: /taskflow/config
   * Type: String
   * Copy snsTopicArn from the console and update the value
   * Copy sqsQueueUrl from the console and update the value
   * Value:

{

"snsTopicArn": "arn:aws:sns:us-east-1:XXX:TaskFlow-Notifications",

"sqsQueueUrl": "https://sqs.us-east-1.amazonaws.com/XXX/TaskFlow-ProcessingQueue"

}

**Step 21: Create EventBridge Rule**

1. Go to EventBridge Console
2. Create rule:
   * Name: TaskFlow-DailyReport
3. Choose Schedule
4. Click Continue in EventBridge Scheduler
5. Choose Recurring schedule
6. Choose Rate-based schedule
7. Type for rate 1
8. Choose days (It shows like this: rate (1 days))
9. Flexible time window: Off
10. Start date and time choose tomorrow (its in date format)
11. For the time (hh:mm) type 00:00
12. Choose Next
13. In Select target choose Lambda
14. Lambda function: TaskFlow-Analytics
15. Next
16. Create schedule

**Step 22: Setup CloudTrail**

1. Go to CloudTrail Console
2. Create trail:
   * Name: TaskFlow-Trail
   * Storage: Create new S3 bucket

**Step 23: Create SQS Processor Lambda**

1. Create a new Lambda function in the console:
2. **Function name**: TaskFlow-SQSProcessor **Runtime**: Python 3.9 **Role**: LabRole
3. **Update the code**, the code is available at TaskFlow-SQSProcessor.py
4. Configure SQS Trigger in Console
5. Go to your lambda console, click at TaskFlow-SQSProcessor
6. Click at Configuration
7. Click at Triggers
8. Add Trigger
   1. Choose SQS
   2. SQS queue: TaskFlow-ProcessingQueue
   3. Choose: Activate trigger, Enable metrics
   4. Batch size: 10
   5. Keep the defaults
   6. Click Add

**Step 24: Update API Gateway**

Add the bulk-import endpoint:

1. Go to API Gateway → TaskFlowAPI
2. Under /tasks resource, create new resource:
   * Resource name: bulk-import
   * Resource path: bulk-import
3. Create POST method:
   * Integration: Lambda Function
   * Function: TaskFlow-TaskHandler
   * Use Lambda Proxy: Yes
   * Authorization: TaskFlowCognitoAuth
4. Deploy API (Actions → Deploy API → Stage: prod)