Day 1 challenge Real Flow

Day 1 Achievement Summary

1. Technical Implementation

```
graph TD
    A[API Gateway] -->|REST API| B[Lambda Function]
    B -->|GetItem| C[DynamoDB]
    D[CloudFormation] -->|Deploys| E[All Resources]
    F[IAM Roles] -->|Permissions| B
```

2. Key Learning Points

1. **Infrastructure as Code**
    - CloudFormation template usage
    - Resource dependency management
    - IAM role configuration
2. **AWS Services Integration**
    - API Gateway setup
    - Lambda function implementation
    - DynamoDB table design
    - IAM role management
3. **Best Practices**
    - Error handling
    - Logging implementation
    - Security considerations
    - Cost optimization
4. **Troubleshooting Skills**

- Permission issues resolution
- API Gateway configuration
- Lambda function testing
- DynamoDB data verification

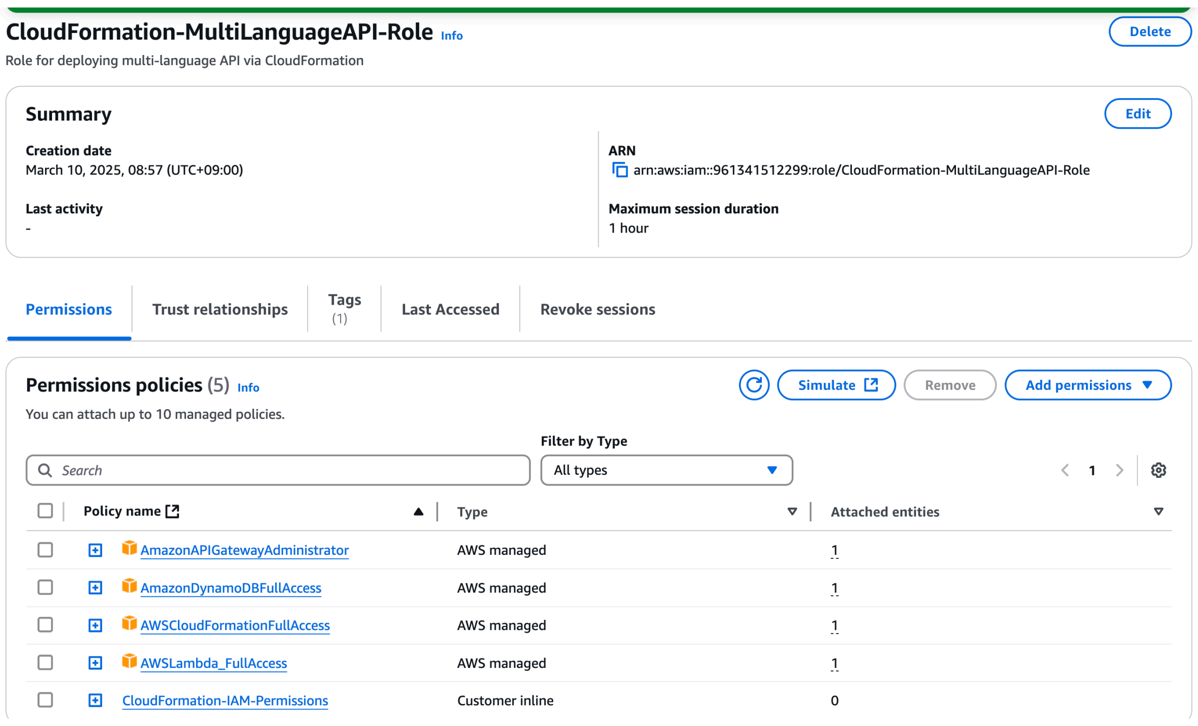
3. Interview-Relevant Experience

```
Technical Skills Demonstrated:
  - Serverless Architecture Design
  - Multi-language Support Implementation
  - Database Integration
  - API Development
  - Security Configuration

Problem-Solving:
  - Debug Permission Issues
  - API Gateway Configuration
  - Unicode Character Handling
  - Error Response Handling

Cost Optimization:
  - Pay-per-request DynamoDB
  - Minimal Lambda memory allocation
  - Serverless architecture
```

4. Ready for Interview Questions Like:

1. "How would you implement a multi-language support system?"
2. "Explain your experience with serverless architecture"
3. "How do you handle permissions and security in AWS?"
4. "Describe your experience with CloudFormation"
5. "How would you troubleshoot API issues?"

1.IAM roles


CloudFormation-MultiLanguageAPI-Role Info

Role for deploying multi-language API via CloudFormation

Summary

Creation date
March 10, 2025, 08:57 (UTC+09:00)

Last activity
-

ARN
arn:aws:iam::961341512299:role/CloudFormation-MultiLanguageAPI-Role

Maximum session duration
1 hour

Permissions | Trust relationships | Tags (1) | Last Accessed | Revoke sessions

Permissions policies (5) Info

You can attach up to 10 managed policies.

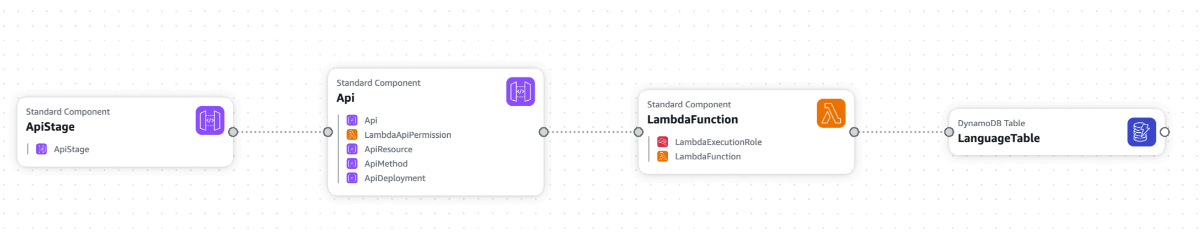
Filter by Type: All types

Policy name	Type	Attached entities
AmazonAPIGatewayAdministrator	AWS managed	1
AmazonDynamoDBFullAccess	AWS managed	1
AWSCloudFormationFullAccess	AWS managed	1
AWSLambda_FullAccess	AWS managed	1
CloudFormation-IAM-Permissions	Customer inline	0

inline policy added:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:PutRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:TagRole",
                "iam:UntagRole"
            ],
            "Resource": [
                "arn:aws:iam::961341512299:role/multi-language-api-stack-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::961341512299:role/multi-language-api-stac
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lambda.amazonaws.com",
                        "apigateway.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetResources",
                "tag:UntagResources",
                "tag:GetTagValues",
                "tag:GetTagKeys",
                "tag:TagResources"
            ],
            "Resource": "*"
        }
    ]
}
```

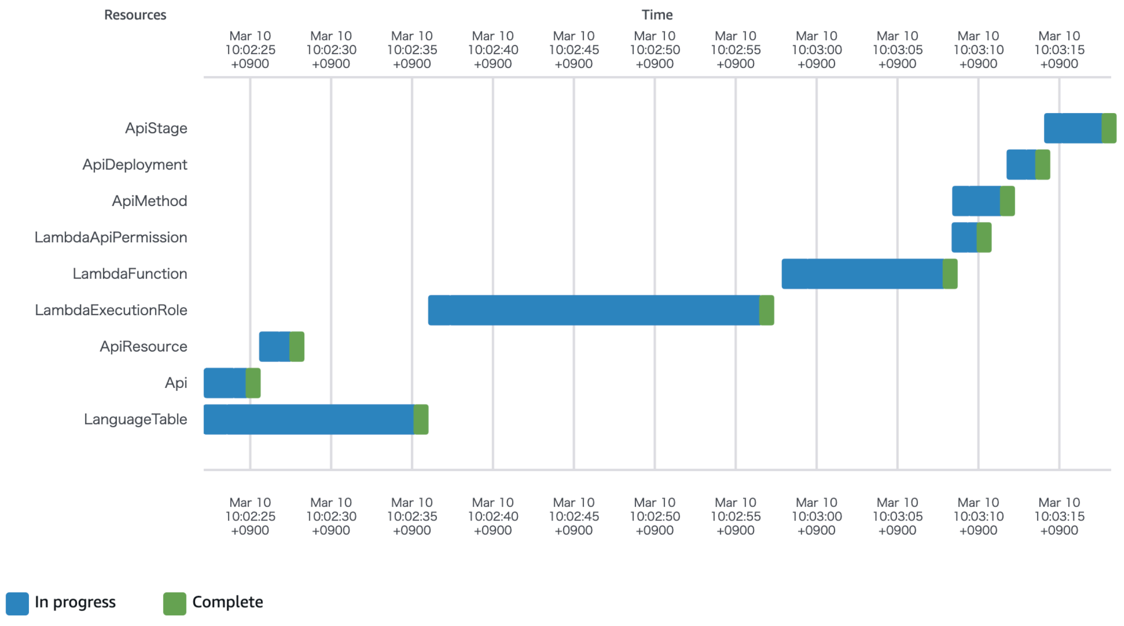
CloudFormation Template:



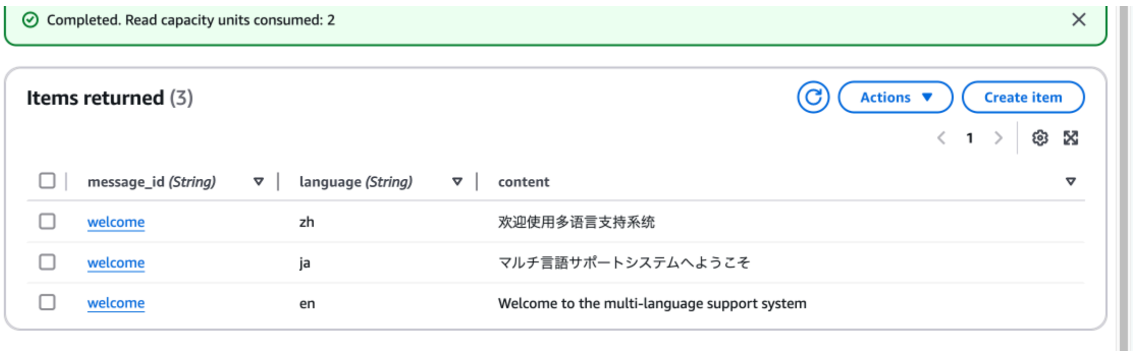
```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Multi-language Support API Stack'

Resources:
  # DynamoDB Table
  LanguageTable:
    Type: 'AWS::DynamoDB::Table'
    Properties:
      TableName: LanguageContent
      BillingMode: PAY_PER_REQUEST
      AttributeDefinitions:
        - AttributeName: message_id
          AttributeType: S
        - AttributeName: language
          AttributeType: S
      KeySchema:
        - AttributeName: message_id
          KeyType: HASH
        - AttributeName: language
          KeyType: RANGE

  # Lambda Execution Role
  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: !Sub '${AWS::StackName}-lambda-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
      Policies:
        - PolicyName: DynamoDBAccess
          PolicyDocument:
```

```
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - dynamodb:GetItem
            Resource: !GetAtt LanguageTable.Arn

  # Lambda Function
  LambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      FunctionName: !Sub '${AWS::StackName}-function'
      Runtime: python3.9
      Handler: index.lambda_handler
      Code:
        ZipFile: |
          import json
          import boto3
          from botocore.exceptions import ClientError
          import logging

          # Setup logging
          logger = logging.getLogger()
          logger.setLevel(logging.INFO)

          def lambda_handler(event, context):
              # Log the incoming event
              logger.info('Event: %s', json.dumps(event))

              try:
                  # Initialize DynamoDB
                  dynamodb = boto3.resource('dynamodb')
                  table = dynamodb.Table('LanguageContent')

                  # Log table name
                  logger.info('Table name: %s', table.table_name)

                  # Get query parameters
                  params = event.get('queryStringParameters', {}) or {}
                  message_id = params.get('message_id', 'welcome')
                  language = params.get('language', 'en')

                  # Log parameters
                  logger.info('Parameters: message_id=%s, language=%s', message

                  if language not in ['en', 'ja', 'zh']:
                      return {
                          'statusCode': 400,
                          'headers': {
                              'Content-Type': 'application/json',
                              'Access-Control-Allow-Origin': '*'
                          },
                          'body': json.dumps({
                              'error': f'Unsupported language: {language}'
                          })
                      }

                  # Try to get item from DynamoDB
                  try:
                      response = table.get_item(
                          Key={
                              'message_id': message_id,
                              'language': language
                          }
                      )
                      logger.info('DynamoDB Response: %s', json.dumps(response))

                      message = response.get('Item', {}).get('content', 'Message not f

                      return {
                          'statusCode': 200,
                          'headers': {
                              'Content-Type': 'application/json',
                              'Access-Control-Allow-Origin': '*'
                          },
                          'body': json.dumps({
                              'message': message,
                              'language': language
                            })
                          }

                        except ClientError as e:
                          logger.error('DynamoDB Error: %s', str(e))
                          return {
                            'statusCode': 500,
                            'headers': {
                              'Content-Type': 'application/json',
                              'Access-Control-Allow-Origin': '*'
                            },
                            'body': json.dumps({
                              'error': 'Database error',
                              'details': str(e)
                            })
                          }

                      except Exception as e:
                        logger.error('General Error: %s', str(e))
                        return {
                          'statusCode': 500,
                          'headers': {
                            'Content-Type': 'application/json',
                            'Access-Control-Allow-Origin': '*'
                          },
                          'body': json.dumps({
                            'error': 'Internal server error',
                            'details': str(e)
                          })
                        }
                  Role: !GetAtt LambdaExecutionRole.Arn
                  Timeout: 10
                  MemorySize: 128

                # Lambda Permission for API Gateway
                LambdaApiPermission:
```

```
  Type: 'AWS::Lambda::Permission'
  Properties:
    FunctionName: !GetAtt LambdaFunction.Arn
    Action: 'lambda:InvokeFunction'
    Principal: 'apigateway.amazonaws.com'
    SourceArn: !Sub 'arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}

# API Gateway
Api:
  Type: 'AWS::ApiGateway::RestApi'
  Properties:
    Name: !Sub '${AWS::StackName}-api'
    EndpointConfiguration:
      Types:
        - REGIONAL

ApiResource:
  Type: 'AWS::ApiGateway::Resource'
  Properties:
    ParentId: !GetAtt Api.RootResourceId
    PathPart: 'message'
    RestApiId: !Ref Api

ApiMethod:
  Type: 'AWS::ApiGateway::Method'
  Properties:
    HttpMethod: GET
    ResourceId: !Ref ApiResource
    RestApiId: !Ref Api
    AuthorizationType: NONE
    Integration:
      Type: AWS_PROXY
      IntegrationHttpMethod: POST
      Uri: !Sub
        - arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functio
        - lambdaArn: !GetAtt LambdaFunction.Arn

ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: ApiMethod
    Properties:
      RestApiId: !Ref Api

  ApiStage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
      DeploymentId: !Ref ApiDeployment
      RestApiId: !Ref Api
      StageName: prod

Outputs:
  ApiEndpoint:
    Description: 'API Endpoint'
    Value: !Sub 'https://${Api}.execute-api.${AWS::Region}.amazonaws.com/pr
  TableName:
    Description: 'DynamoDB Table Name'
    Value: !Ref LanguageTable
  LambdaFunction:
    Description: 'Lambda Function Name'
    Value: !Ref LambdaFunction
```



add items in DynamoDB:


Completed. Read capacity units consumed: 2

Items returned (3)

message_id (String)	language (String)	content
welcome	zh	欢迎使用多语言支持系统
welcome	ja	マルチ言語サポートシステムへようこそ
welcome	en	Welcome to the multi-language support system

updated lambda function (Not necessary,CF template has been updated):

```
import json
import boto3
from botocore.exceptions import ClientError
import logging

# Setup logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Log the incoming event
    logger.info('Event: %s', json.dumps(event))

    try:
        # Initialize DynamoDB
        dynamodb = boto3.resource('dynamodb')
        table = dynamodb.Table('LanguageContent')

        # Log table name
        logger.info('Table name: %s', table.table_name)

        # Get query parameters
```

```
params = event.get('queryStringParameters', {}) or {}
message_id = params.get('message_id', 'welcome')
language = params.get('language', 'en')

# Log parameters
logger.info('Parameters: message_id=%s, language=%s', message_id, la

if language not in ['en', 'ja', 'zh']:
    return {
        'statusCode': 400,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'error': f'Unsupported language: {language}'
        })
    }

# Try to get item from DynamoDB
try:
    response = table.get_item(
        Key={
            'message_id': message_id,
            'language': language
        }
    )
    logger.info('DynamoDB Response: %s', json.dumps(response))

    message = response.get('Item', {}).get('content', 'Message not found')

    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
                'message': message,
                'language': language
            })
        }

    except ClientError as e:
        logger.error('DynamoDB Error: %s', str(e))
        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'error': 'Database error',
                'details': str(e)
            })
        }

except Exception as e:
    logger.error('General Error: %s', str(e))
    return {
        'statusCode': 500,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'error': 'Internal server error',
            'details': str(e)
        })
    }
```

Test function:event json

```
{
  "queryStringParameters": {
    "language": "en",
    "message_id": "welcome"
  }
}
```

API endpoint Test:

1. English:

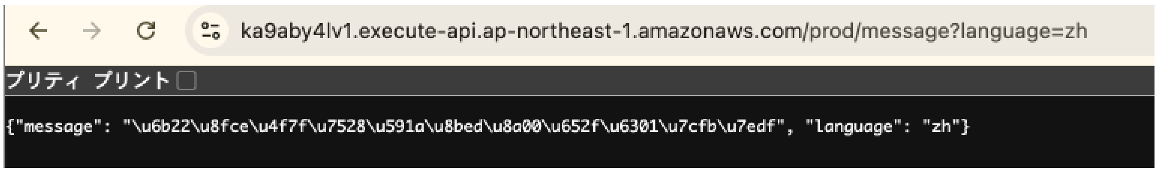
```
<https://ka9aby4lv1.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=en>
```

1. Japanese:

```
<https://ka9aby4lv1.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=ja>
```

1. Chinese (already working):

```
<https://ka9aby4lv1.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=zh>
```

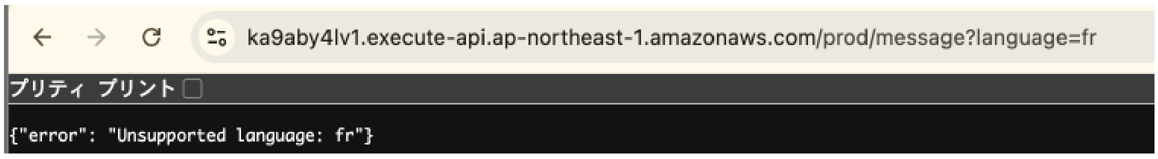

ka9aby4lv1.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=zh
プリティ プリント

```
{"message": "\u6b22\u8fce\u4f7f\u7528\u591a\u8bed\u8a00\u652f\u6301\u7cfb\u7edf", "language": "zh"}
```

You can also test error handling:

```
<https://ka9aby4lv1.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=fr>
```

Should return a 400 error as French is not supported.


ka9aby4lv1.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=fr
プリティ プリント

```
{"error": "Unsupported language: fr"}
```

Appdendix

CF template details:

Let's break down both the CloudFormation template and the inline policy for a complete understanding.

1. CloudFormation Template Breakdown

A. Resource Organization

```
graph TD
    A[CloudFormation Template] --> B[Storage Layer]
    A --> C[Compute Layer]
    A --> D[API Layer]
    A --> E[Security Layer]
    B --> B1[DynamoDB Table]
    C --> C1[Lambda Function]
    D --> D1[API Gateway]
    E --> E1[IAM Roles/Policies]
```

B. Detailed Component Analysis

1. Storage Layer (DynamoDB)

```
LanguageTable:
  Type: 'AWS::DynamoDB::Table'
  Properties:
    TableName: LanguageContent
    BillingMode: PAY_PER_REQUEST  # Cost-effective, no capacity planning
needed
    AttributeDefinitions:
      - AttributeName: message_id  # Partition key
        AttributeType: S
      - AttributeName: language    # Sort key
        AttributeType: S
    KeySchema:
      - AttributeName: message_id
        KeyType: HASH              # Primary key
      - AttributeName: language
        KeyType: RANGE             # Sort key
```

Key Points:

- Composite key for efficient queries
- PAY_PER_REQUEST for cost optimization
- No provisioned capacity needed

2. Compute Layer (Lambda)

```
LambdaFunction:
  Type: 'AWS::Lambda::Function'
  Properties:
    FunctionName: !Sub '${AWS::StackName}-function'
    Runtime: python3.9
    Handler: index.lambda_handler
    Code:
      ZipFile: |
        # Lambda function code here
    Role: !GetAtt LambdaExecutionRole.Arn
    Timeout: 10
    MemorySize: 128  # Minimum memory for cost efficiency
```

Key Points:

- Inline code deployment

- Minimal memory allocation
- 10-second timeout
- Role attached via GetAtt

3. API Layer (API Gateway)

```
Api:
  Type: 'AWS::ApiGateway::RestApi'
  Properties:
    Name: !Sub '${AWS::StackName}-api'
    EndpointConfiguration:
      Types:
        - REGIONAL    # Regional endpoint for better latency

ApiResource:
  Type: 'AWS::ApiGateway::Resource'
  Properties:
    ParentId: !GetAtt Api.RootResourceId
    PathPart: 'message'
    RestApiId: !Ref Api

ApiMethod:
  Type: 'AWS::ApiGateway::Method'
  Properties:
    HttpMethod: GET
    ResourceId: !Ref ApiResource
    RestApiId: !Ref Api
    AuthorizationType: NONE
    Integration:
      Type: AWS_PROXY  # Lambda proxy integration
      IntegrationHttpMethod: POST
      Uri: !Sub arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/
functions/${LambdaFunction.Arn}/invocations
```

Key Points:

- Regional endpoint configuration
- Lambda proxy integration

- No authentication (for simplicity)

4. Security Layer (IAM)

```
LambdaExecutionRole:
  Type: 'AWS::IAM::Role'
  Properties:
    RoleName: !Sub '${AWS::StackName}-lambda-role'
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Policies:
      - PolicyName: DynamoDBAccess
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - dynamodb:GetItem
              Resource: !GetAtt LanguageTable.Arn
```

2. Inline Policy Breakdown for CloudFormation Role

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:PutRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:TagRole",
                "iam:UntagRole"
            ],
            "Resource": [
                "arn:aws:iam::961341512299:role/multi-language-api-stack-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::961341512299:role/multi-language-api-st
ack-*",
            "Condition": {
                "StringEquals": {
                  "iam:PassedToService": [
                    "lambda.amazonaws.com",
                    "apigateway.amazonaws.com"
                  ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetResources",
                "tag:UntagResources",
                "tag:GetTagValues",
                "tag:GetTagKeys",
                "tag:TagResources"
            ],
            "Resource": "*"
        }
```

```
  ]
}
```

Policy Sections Explained:

1. IAM Role Management

```
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateRole",
    "iam:DeleteRole",
    // ... other actions
  ],
  "Resource": [
    "arn:aws:iam::961341512299:role/multi-language-api-stack-*"
  ]
}
```

- Allows creation and management of IAM roles
- Restricted to roles with specific prefix
- Limited to your account ID

2. PassRole Permission

```
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::961341512299:role/multi-language-api-stack-
*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": [
        "lambda.amazonaws.com",
        "apigateway.amazonaws.com"
      ]
    }
    }
}
```

- Allows passing roles to specific services
- Limited to Lambda and API Gateway
- Includes condition for security

3. Tagging Permissions

```
{
    "Effect": "Allow",
    "Action": [
        "tag:GetResources",
        // ... other tag actions
    ],
    "Resource": "*"
}
```

- Required for CloudFormation resource tagging
- Applies to all resources (needed for CF operation)

Manually without CloudFormation

Manual Deployment Process - Multi-language API

Step 1: Create DynamoDB Table

```
graph TD
    A[DynamoDB Console] --> B[Create Table]
    B --> C[Configure Settings]
    C --> D[Create]
```

1. Go to DynamoDB Console
2. Click "Create table"
3. Configure:

```
Table name: LanguageContent-Manual
Partition key: message_id (String)
Sort key: language (String)
Table settings:
  - Default settings
  - Customize settings:
    - Capacity mode: On-demand
```

1. Click "Create table"

Step 2: Create IAM Role for Lambda

```
graph TD
  A[IAM Console] --> B[Create Role]
  B --> C[Lambda Use Case]
  C --> D[Add Permissions]
  D --> E[Create Role]
```

1. Go to IAM Console
2. Click "Roles" → "Create role"
3. Select Use Case: "Lambda"
4. Add policies:
    - `AWSLambdaBasicExecutionRole`
5. Create custom policy:
    - Click "Create policy"
    - JSON tab:

```
{
  "Version": "2012-10-17",
  "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/LanguageContent-Manua
l"
        }
    ]
}
```

1. Name: `DynamoDB-GetItem-Manual`
2. Add this policy to the role
3. Role name: `lambda-multilang-manual-role`
4. Create role

Step 3: Create Lambda Function

1. Go to Lambda Console
2. Click "Create function"
3. Configure:

```
Function name: multilang-api-manual
Runtime: Python 3.9
Architecture: x86_64
Permissions:
    - Use existing role
    - Select: lambda-multilang-manual-role
```

1. Click "Create function"
2. Add code:

```
import json
import boto3
from botocore.exceptions import ClientError
import logging
```

```
# Setup logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Log the incoming event
    logger.info('Event: %s', json.dumps(event))

    try:
        # Initialize DynamoDB
        dynamodb = boto3.resource('dynamodb')
        table = dynamodb.Table('LanguageContent-Manual')

        # Log table name
        logger.info('Table name: %s', table.table_name)

        # Get query parameters
        params = event.get('queryStringParameters', {}) or {}
        message_id = params.get('message_id', 'welcome')
        language = params.get('language', 'en')

        # Log parameters
        logger.info('Parameters: message_id=%s, language=%s', message_id,
language)

        if language not in ['en', 'ja', 'zh']:
            return {
                'statusCode': 400,
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({
                    'error': f'Unsupported language: {language}'
                })
            }
        # Try to get item from DynamoDB
        try:
            response = table.get_item(
                Key={
                    'message_id': message_id,
                    'language': language
                }
            )
            logger.info('DynamoDB Response: %s', json.dumps(response))

            message = response.get('Item', {}).get('content', 'Message not foun
d')

            return {
                'statusCode': 200,
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({
                    'message': message,
                    'language': language
                })
            }

        except ClientError as e:
            logger.error('DynamoDB Error: %s', str(e))
            return {
                'statusCode': 500,
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({
                    'error': 'Database error',
                    'details': str(e)
                })
            }
```

```
    except Exception as e:
        logger.error('General Error: %s', str(e))
        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'error': 'Internal server error',
                'details': str(e)
            })
        }
```

1. Click "Deploy"

Step 4: Create API Gateway

1. Go to API Gateway Console
2. Click "Create API"
3. Choose REST API (not private)
4. Configure:

```
API name: MultiLang-Manual-API
Description: Manual version of multi-language API
Endpoint Type: Regional
```

1. Click "Create API"
2. Create Resource:
    - Click "Actions" → "Create Resource"
    - Resource Name: message
    - Resource Path: /message
    - Click "Create Resource"
3. Create Method:

- Click "Actions" → "Create Method"
- Select GET
- Configure:
  - Integration type: Lambda Function
  - Lambda Function: multilang-api-manual
  - Use Lambda Proxy integration: Yes
- Click "Save"
- Click "OK" to give permission

4. Deploy API:
   - Click "Actions" → "Deploy API"
   - Stage name: prod
   - Click "Deploy"

Step 5: Add Test Data to DynamoDB

1. Go to DynamoDB Console
2. Select "LanguageContent-Manual" table
3. Click "Explore table items"
4. Add items:

Item 1:

```
{
  "message_id": "welcome",
  "language": "en",
  "content": "Welcome to the multi-language support system"
}
```

Item 2:

```
{
  "message_id": "welcome",
  "language": "ja",
    "content": "マルチ言語サポートシステムへようこそ"
}
```

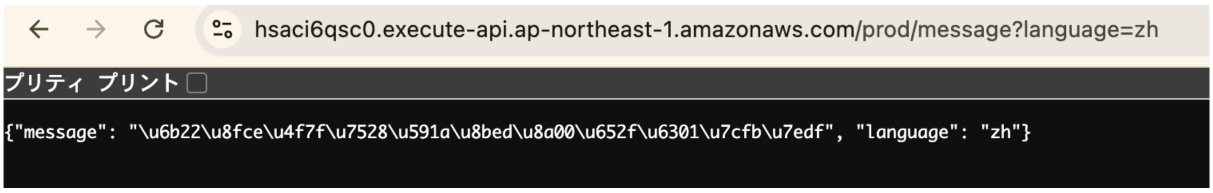
Item 3:

```
{
    "message_id": "welcome",
    "language": "zh",
    "content": "欢迎使用多语言支持系统"
}
```

Step 6: Test the API

1. Get your API URL from API Gateway Console:
    - Click on "Stages"
    - Click on "prod"
    - Copy "Invoke URL"
2. Test endpoints:

```
# English
curl "[YOUR_API_URL]/message?language=en"

# Japanese
curl "[YOUR_API_URL]/message?language=ja"

# Chinese
curl "[YOUR_API_URL]/message?language=zh"

# Error case
curl "[YOUR_API_URL]/message?language=fr"
```



```
hsaci6qsc0.execute-api.ap-northeast-1.amazonaws.com/prod/message?language=zh
```

```
{"message": "\u6b22\u8fce\u4f7f\u7528\u591a\u8bed\u8a00\u652f\u6301\u7cfb\u7edf", "language": "zh"}
```