Ace It - AI Study Assistant Deployment Guide

For more information, please visit AceIT Repository (https://github.com/UBC-CIC/AceIT-ECE-Capstone).

Deploying & Running Ace It on AWS

Pre-Requisites

To deploy Ace It, you will need to first have the following pre-requisites:

1. A Canvas LMS instance where you have the Admin role
    - If you need to first set up a Canvas instance, please refer to the Canvas documentation here (https://github.com/instructure/canvas-lms/wiki/Production-Start)
2. An AWS account with appropriate permissions for deployment

Instructions

Deploying Ace It on your AWS account requires minor configuration in several areas across infrastructure, Canvas LMS, backend, and frontend.

Canvas LMS Configuration

To integrate this project with **Canvas LMS**, follow these steps:

1. AceIt Canvas Account creation

- It is recommended to create a Canvas Admin Account for AceIt integration use only.

1. Log in to your **Canvas LMS** instance as an **Admin**.
2. Navigate through **Admin** > **Site Admin** > **Settings** > **Admins**
3. Click **+ Account Admins**
4. Fill in your account info.
5. Confirm the account details and Click **Continue** > **OK**

2. Access Token generation

An access token is required to retrieve course content from Canvas.

1. Log in to your **AceIt** Canvas Admin Account
2. Navigate to **Account** > **Settings**.
3. Scroll down to the **Approved Integrations** section.
4. Click **+ New Access Token**.
5. Enter a purpose (e.g., "AceIt Integration") and click **Generate Token**.
6. Copy the generated token and store it securely (this token will not be shown again).

3. Developer Key Generation

To allow authentication with the Canvas account, create a **Developer Key**.

1. Navigate to **Admin** > **Developer Keys**.
2. Click **+ Developer Key** > **API Key**.
3. Fill in the required details:
    - **Key Name**: (e.g., "AceIt Key")
    - **Redirect URIs**: The CloudFront Distribution domain name set up in AWS Configuration steps.
    - **Scopes**: Select appropriate API permissions for accessing course data. (Disable scope to allow access to all endpoints)
4. Click **Save Key** and toggle it **ON**.
5. Copy the **Client ID** and **Client Secret**.

AWS Configuration

To deploy Ace It on AWS, you'll need an AWS account with the appropriate permissions. This section covers two deployment methods:

- Method 1: Using GitHub Actions for CI/CD deployment.
- Method 2: Manual deployment without using GitHub Actions.

Pre-Requisites

- AWS Account: Ensure you have an AWS account with administrative access.
- AWS Bedrock Request Access to: (In AWS Console > AWS Bedrock)
    - Llama 3.3
    - Amazon Text Embeddings Titan v2
- Clone this Github repository.

Method I. Using Github Action [Recommended]

1. Create an IAM role that GitHub Actions can assume to deploy resources on AWS. 1.1 Go to AWS Console > IAM > Create role, make sure you are in the desired region. 1.2 Under Select trusted entity, choose Web identity > Add Identity provider 1.3 Select OpenID Connect, Provider URL will be `token.actions.githubusercontent.com`, click Add Provider 1.4 Add the Identity Provider you just created, set Audience to be `sts.amazon.aws.com`. 1.5 Fill in the Github organization 1.6 Add permissions. For

simplicity, we will attach `AdministratorAccess` to allow GitHub Action to have full access to AWS services. 1.7 Give this role a name, and create this role. 1.8 Go to the Github repository > Settings > Secrets and Variables > Actions 1.9 Create secrets for:

   - AWS_ROLE_ARN: The ARN of the IAM Role created.
   - AWS_REGION: Your preferred AWS region (e.g., us-west-2).
   - AWS_ACCOUNT_ID: Your AWS Account ID. 1.10 Open the github workflows in .github/workflows/deploy.yml, replace the value of `role-to-assume` and `aws-region` to be `${{ secrets.AWS_ROLE_ARN }}` and `${{ secrets.AWS_REGION }}` respectively.

2. Create an S3 bucket to host the frontend files. 2.1 Go to AWS Console > S3 > Create bucket > General Purpose, make sure you are in the desired region. 2.2 Give a name to the bucket, this is for the front end, so uncheck `Block all public access` 2.3 Bucket Policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "PublicReadGetObject",
          "Effect": "Allow",
          "Principal": "*",
          "Action": "s3:GetObject",
          "Resource": "arn:aws:s3:::myfrontendbucket3/*"
      }
  ]
}
```

3. Set Up CloudFront 3.1 Go to AWS Console > CloudFront > Create distribution, make sure you are in the desired region. 3.2 Create Distribution:
   - Origin Domain Name: Select the S3 bucket created in step 2.
   - Viewer Protocol Policy: Redirect HTTP to HTTPS
   - Allowed HTTP Methods: GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE
   - Cache Policy: Managed-CachingOptimized
   - Origin Request Policy: CORS-S3Origin
   - Response headers policy name: Managed-CORS-With-Preflight 3.3 After creating distribution, Copy the Distribution domain name. It'll be used for Access-Control-Allow-Origin.
4. Modify .github/workflows/deploy.yml:
   - replace `role-to-assume` to be `${{ secrets.AWS_ROLE_ARN }}` `aws-region`: `${{ secrets.AWS_REGION }}`

Method II. No Github Action

1. follow instructions on this AWS Documentation (https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html) to install AWS CLI and configure local AWS Credentials.

2. Create an S3 bucket to host the frontend files. 2.1 Go to AWS Console > S3 > Create bucket > General Purpose, make sure you are in the desired region. 2.2 Give a name to the bucket, this is for the front end, so uncheck `Block all public access` 2.3 Upload the all files in frontend to the S3 bucket created. 2.4 Bucket Policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "PublicReadGetObject",
          "Effect": "Allow",
          "Principal": "*",
          "Action": "s3:GetObject",
          "Resource": "arn:aws:s3:::myfrontendbucket3/*"
      }
  ]
}
```

3. Set Up CloudFront, follow the steps in Method I step 3.

Backend Configuration

1. Due to security considerations, it is the best practice to manually create Canavs Secrets. 1.1 Go to AWS Console > Secrets Manager > Store a new secret 1.2 Select secret type = Other type of secret 1.3 Under the Secret Value - Key/Value, enter:

Secret key	Secret value
apiKeyId	The API Developer Key Id displayed on Canvas
apiKey	The API Developer Key obtained from Canvas
baseURL	The base URL of the Canvas server
redirectURI	The CloudFront Distribution domain name set up in AWS Configuration steps
adminAccessToken	The Canvas Admin Access Token

2. Give this Secret a name. If the name is other than " CanvasSecrets", please also change the secret name in get_canvas_secret.py. Replace `secret_name` with the name of your secret (e.g. "CanvasSecrets")
3. Once the necessary secrets have been configured in AWS Secrets Manager, you can deploy the backend code. There are two methods for deployment:

- Method I. Using Github Action [Recommended] 3.1.1 Create a new GitHub repository (if not already done): Go to GitHub → New Repository → Set the repository name (e.g., AceIT-ECE-Capstone). 3.1.2 Commit and push your cloned repository to GitHub. 3.1.3 Ensure that GitHub Secrets are properly set up in AWS Configuration step 3.1.4 Trigger the GitHub Actions deployment - Push any changes to main to trigger the deployment workflow 3.1.5 If successful, the backend Lambda functions and infrastructure should be deployed. You can check the deployed stack url in GitHub Repository > Actions > Select the latest deployment workflow. 3.1.6 To prefix the deployed resources the github actions will deploy, simply modify the github action workflow file `deploy.yml` line 59 to be `npx cdk deploy --require-approval never --context env_prefix=prefixlikedev`

- Method II. Manual Deployment Without GitHub Actions 3.2.1 If you prefer not to use GitHub Actions, follow these steps to deploy the backend manually using AWS CDK. 3.2.2 Install AWS CDK, python dependencies by running `npm install -g aws-cdk`, `pip install -r requirements.txt` and `pip install -r requirements-dev.txt`in terminal and make sure the working directory is at the root, i.e. ACEIT-ECE-CAPSTONE. 3.2.3 Make sure you've completed steps in AWS Configuration Method II to set up your AWS Credentials 3.2.4 Type in terminal `cdk bootstrap`, then type `cdk deploy --require-approval never`. If you want to prefix resources for development and production, simply type `cdk deploy --require-approval never --context env_prefix=prefixlikedev` in the terminal instead. 3.2.5 If successful, there will be a deployed stack url displayed in your terminal. these will be your backend function urls.

4. After deployment is successful, record the stack url (e.g. https://something.execute-api.us-west-2.amazonaws.com/prod/ (https://something.execute-api.us-west-2.amazonaws.com/prod/)). This can be obtained by checking the `Deploy Stack` step if deploying using github action, or by checking the result of step 3.2.5 for no github action method. Replace the `VITE_REACT_APP_API_URL`'s value in frontend/.env with your actual stack url. Also modify the `VITE_REACT_APP_HOST_URI`to your cloud distribution domain name, and `VITE_REACT_APP_CANVAS_URL` to be your canvas url. Then, modify the `lambda/utils/construct_response.py`'s `Access-Control-Allow-Origin` value to be your cloud distribution domain name. Save and deploy again. Now if you visit the Distribution domain name from cloudfront, you can visit AceIt!

- Important:

  > *Please note that due to aws resource name constraints, the prefix should only contain lowercase letters. First time deploying will take ~25 minutes, please be patient.*

Frontend Configuration

There are two key types of frontend configuration. They are as follows:

1. Configure theme and branding in `frontend/theme.ts`
2. Configure API destinations in `frontend/.env`

Further details on what to configure in each of these files can be found within the files themselves.