🧩 Building CI/CD Pipeline for IBM App Connect Enterprise (ACE) on AWS using GitHub Actions

Step 0: Pre-requisites

Before you begin, make sure you have the following ready:

✅ **Software and Accounts**

- **Docker Desktop installed and running on your local system.**
- **GitHub account and repository created.**
- **AWS account with permissions for ECR (Elastic Container Registry) and ECS (Elastic Container Service).**
- **Windows system with administrative rights (for AWS CLI installation).**

✅ **Files and Project**

- **Your IBM ACE BAR file (e.g., http_app.bar).**
- **Dockerfile to build the ACE runtime container.**
- **GitHub Actions workflow file (ace-ci.yml).**
- **Python test files (optional for unit testing).**

✅ **Ports and Runtime**

- **Ensure your ACE project listens on port 5000 (or 7600 in this example).**
- **Keep your AWS Access Key and Secret Key handy.**

Step 1: Develop Your IBM ACE Application

- Open **IBM App Connect Enterprise Toolkit**.
- Create or import your **integration project**.
- Build and test your integration flows:
  - Message flows
  - Subflows
  - Mappings
  - MQ, HTTP, or SOAP nodes
- Test flows locally in the toolkit.

Step 2: Package the Application (Create BAR File)

1. In ACE Toolkit:
    - Right-click your project → Export → BAR file
    - Example output: http_app.bar
2. This BAR file includes:
    - Message flows, Libraries, Configuration data

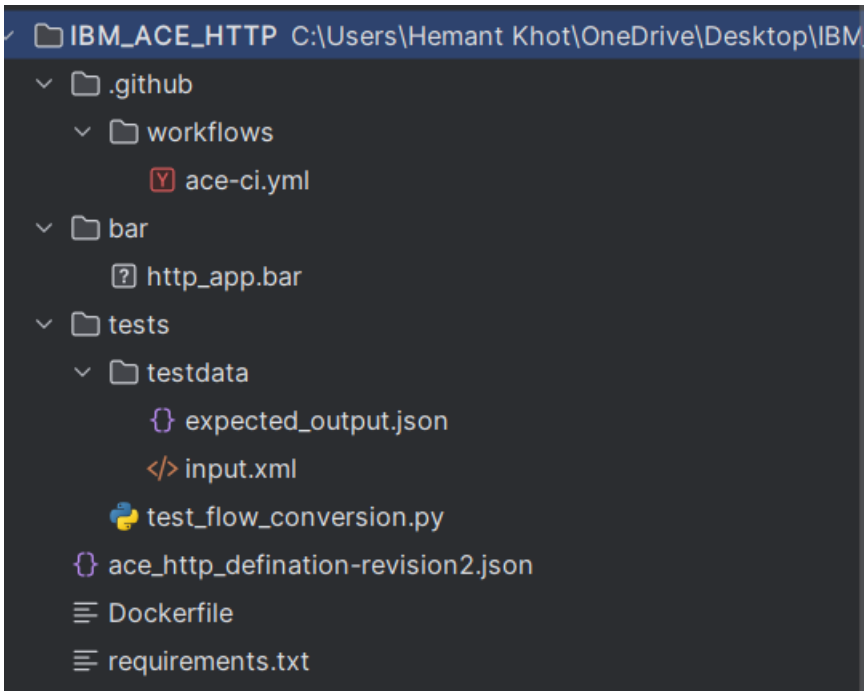
Step 3: Download IBM ACE Docker Image

- IBM provides official ACE runtime images with ACE pre-installed. Use Docker to pull:

docker pull ibmcom/ace:11.0.0.6.3-amd64

No need to manually install ACE. This image has everything ready.

Step 4: Create a Folder for struture Setup



ace-ci.yml: script for performing tha steps

http_bar.bar: ACE application which you want to deploy

tests/testdata: having you input and output data which you want to test after every deployment.

Dockerfile: fecth the the details related to IBM ACE

Requirement.txt : download required python lib.

Step 5: Write the Dockerfile

This Dockerfile tells Docker how to prepare the ACE runtime and deploy your BAR file.

```
# Base image: IBM ACE v11 (already downloaded locally)
FROM ibmcom/ace:11.0.0.6.3-amd64

# Accept license and set environment variables
ENV LICENSE=accept
ENV LANG=en_US.UTF-8
ENV MQSI_BASE_DIRECTORY=/opt/ibm/ace-11


# Copy your BAR file into the image
COPY bar/http_app.bar /home/aceuser/initial-config/bars/

# Optional: Copy custom server config
# COPY server.conf.yaml /home/aceuser/initial-config/

# Switch to non-root user
USER aceuser

# Start Integration Server
CMD ["IntegrationServer", "--name", "MyIntegrationServer2", "--work-dir", "/home/aceuser/ace-server"]
```

Notes:

- **Replace ace_on_aws.bar with your actual BAR file name.**
- **--work-dir is where the integration server runs inside the container.**
- **Optional server.conf.yaml allows you to provide additional configuration for the server.**

Step 6: Log in to AWS Console

1. Open this link in a browser:
   Login AWS Console
2. Enter your **AWS username and password**.
3. Switch the region (top-right) to **us-east-1 (N. Virginia)**.
    - AWS services must be consistent in the same region for your deployment.

Explanation: Logging in lets you access AWS services like ECR (Elastic Container Registry) and ECS (Elastic Container Service). The region determines where your resources will be hosted.

Step 7: Create ECR Repository

1. In AWS console, search **ECR** → click **Repositories** → **Create repository**.
2. Name: flask_loan_app → leave default settings → **Create**.

Explanation: ECR stores your Docker images. This repository will hold your Flask app image so ECS can pull it.

Step 8: Create IAM Access Keys

1. Search **IAM** → Users → click your existing username.
2. Go to **Security credentials** → **Create access key**.
3. Save the **Access Key ID** and **Secret Access Key** securely.

Explanation: These credentials allow AWS CLI to access your AWS account programmatically. Without them, you cannot push images or run ECS tasks from CLI.

Step 9: Install AWS CLI (Windows)

Option A: MSI Installer

Download and install AWS CLI

msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi

Or silent installation

msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi /qn

Verify installation:

aws --version

Example: aws-cli/2.27.41 Python/3.11.6 Windows/10 exe/AMD64

If aws command not recognized:

1. Windows Search → "Edit the system environment variables" → Environment Variables → Edit Path.
2. Add AWS CLI path (e.g., C:\Program Files\Amazon\AWSCLIV2\).

Explanation: AWS CLI allows you to interact with AWS services from the terminal for automation, building images, pushing to ECR, and configuring ECS.

If still didn't get then use below command:

pip install awscli

Step 10: Configure AWS CLI

Open PowerShell/CMD:

aws configure

Enter your keys and region

AWS Access Key ID [None]: <Your_Access_Key_ID>

AWS Secret Access Key [None]: <Your_Secret_Access_Key>

Default region name [None]: us-east-1 (If already present then Hit Enter)

Default output format [None]: Hit Enter

Explanation: This saves your credentials locally and lets AWS CLI run commands on your account.

Step 11: Build and Push Docker Image to ECR

Click on ECR repo where you have created and then click on View Push Commands.

And the Run the commands one by one

Step 12: Create ECS Cluster

1. AWS Console → Search **ECS** → Click **Get Started**.
2. Create a **Fargate cluster** → Name: loan_app_cluster → leave defaults → **Create**.

Explanation: ECS Fargate allows you to run Docker containers without managing servers. A cluster is a logical grouping of tasks/services.

Step 13: Create ECS Task Definition

1. ECS → Task Definitions → **Create new Task Definition** → Select **FARGATE**.
2. Task Definition Name: ace_http_task.
3. Container settings:

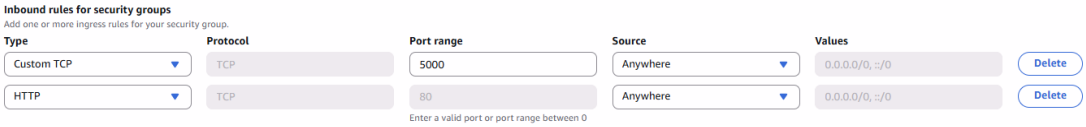
   From ECR section copay the URI from latest and paste in ECS -→ Container --→ image URI

   - Name: ace_http_container
   - Port mapping: **7600** (Flask default)
4. Click **Create** → open the new task definition.

Explanation: Task definition describes the container, image, resources, and ports ECS will run.

Step 14: Run ECS Task

1. ECS → Task Defination /cluster→ Defination name → Deploy drop down → **Create New service**.
2. Launch type: **FARGATE**, select ace_http_task.
3. Networking:
    - Assign public IP: **ENABLED**
    - Security group: create new → allow inbound rules:
        - HTTP → Port 80 → Source: Anywhere
        - Custom TCP → Port 7600 → Source: Anywhere



4. Click **Run Task** → task status should be **RUNNING**.

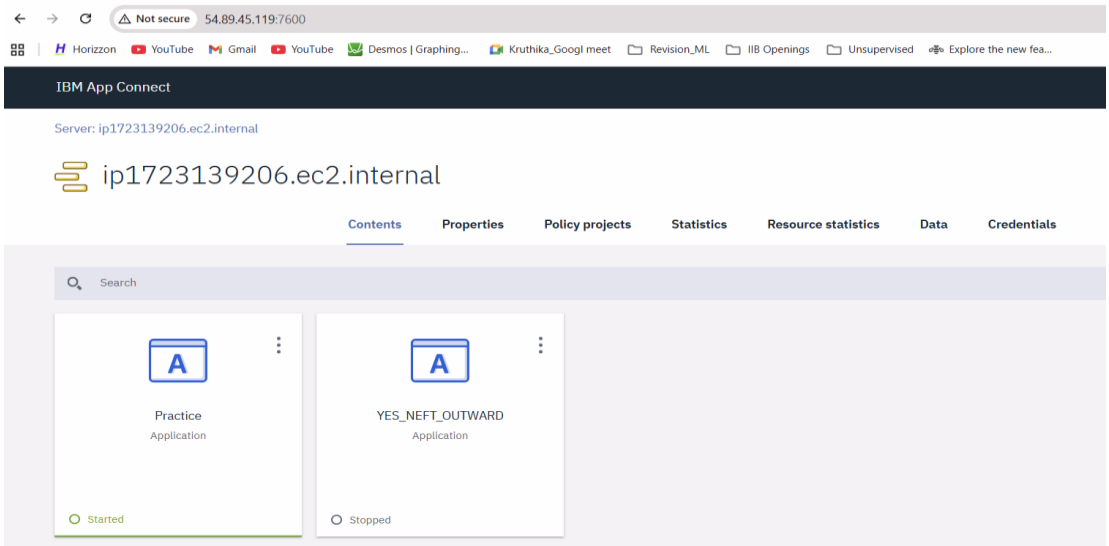
Explanation: Running a task launches the container in ECS, exposes it to the internet via security group rules.

Step 15: Test Your Application deploy on AWS

1. Go to the running task → check **Public IP**.
2. Open browser → http://<PUBLIC_IP>:7600/

Example: http://18.206.12.187:7600/

Explanation: Your Flask app is now live on ECS Fargate and can be accessed via public IP.

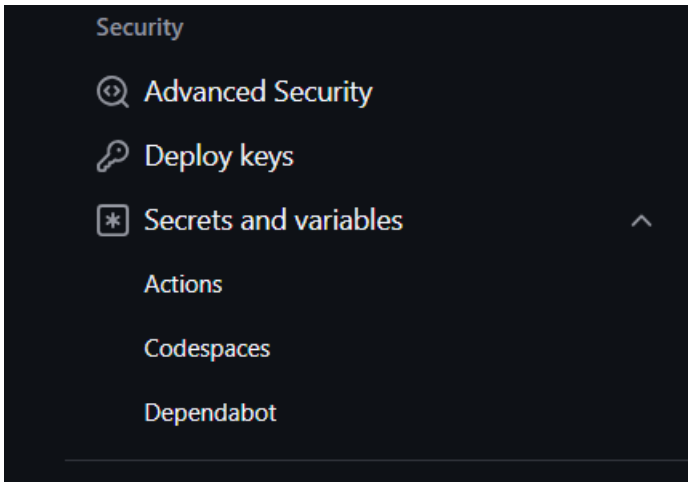


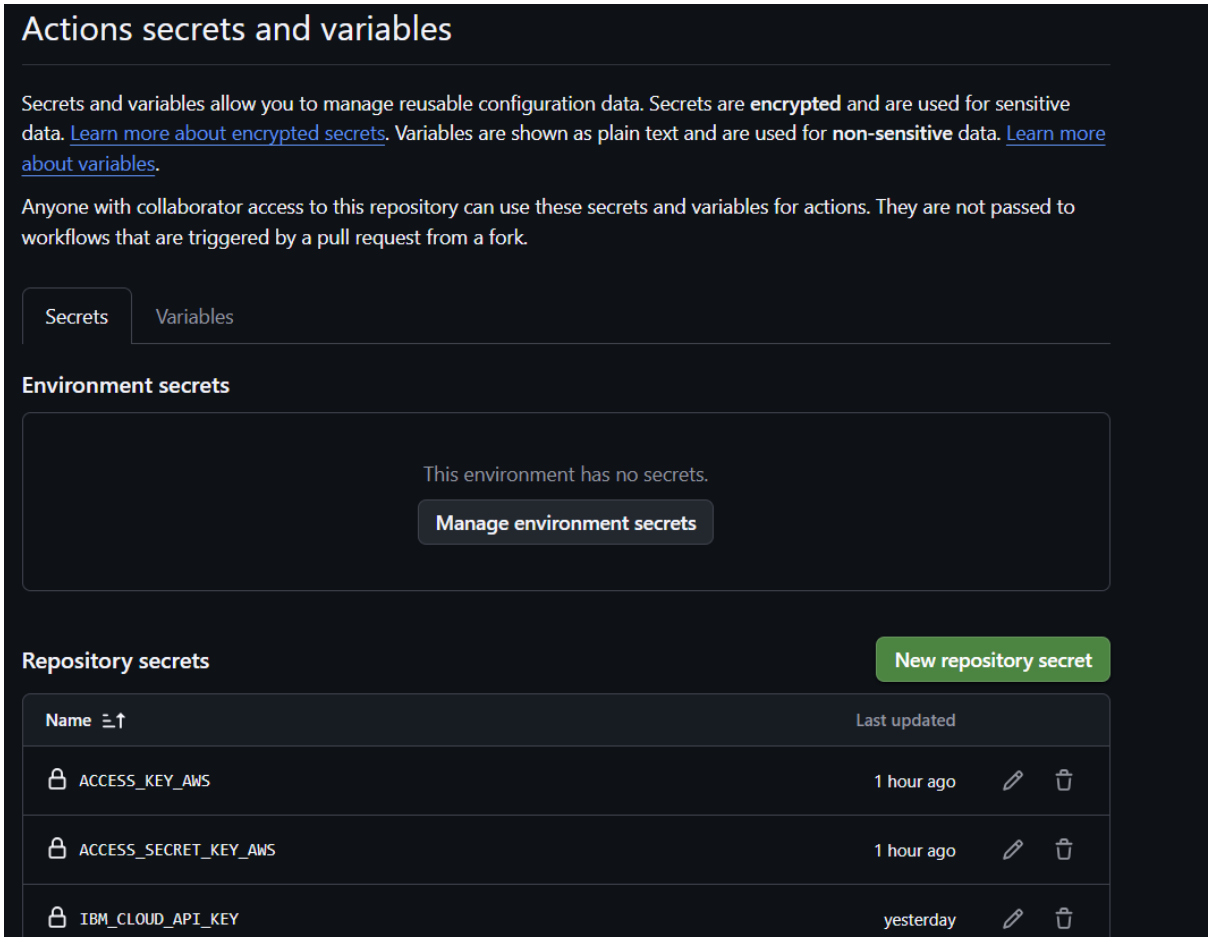
Step 16: Add GitHub Secrets

Go to:
Repo → Settings → Secrets and variables → Actions

Add the following:

Put the same credentials which put step 10.





Step 17: Create GitHub Actions CI/CD Workflow (ace-ci.yml)

Below is the complete code of .yml file Ignore START

**********START**********

```
name: IBM ACE CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  # ------------------------------------------------
  # 🧱 Job 1: Build and Test IBM ACE Application
  # ------------------------------------------------
  build-ace:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v5

      - name: Set up JDK
        uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'temurin'

      - name: Pull IBM ACE Docker image
        run: docker pull ibmcom/ace:11.0.0.6.3-amd64

      - name: Verify existing BAR file
        run: |
          echo "Checking for prebuilt BAR file..."
          ls -lh ${{ github.workspace }}/bar/http_app.bar
          if [ ! -f "${{ github.workspace }}/bar/http_app.bar" ]; then
            echo "❌ BAR file not found in bar/ directory!"
            exit 1
          else
            echo "✅ BAR file found: bar/http_app.bar"
          fi


      - name: Upload BAR artifact
        uses: actions/upload-artifact@v4
        with:
```

```
      name: http_app
      path: bar/http_app.bar

# ---------------------------------------------
# 🧪 Job 2: Run Python Unit Tests
# ---------------------------------------------
test-python:
  needs: build-ace
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v5

    - name: Set up Python 3
      uses: actions/setup-python@v5
      with:
        python-version: 3.13

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install pytest requests

    - name: Start ACE container for testing
      run: |
        docker build -t ace-test .
        docker run -d -p 7600:7800 --name ace-container ace-test
        sleep 25

    - name: Wait for ACE to be ready
      run: |
        echo "Waiting for ACE container to start..."
        for i in {1..40}; do
          if curl -s http://localhost:7600/first > /dev/null; then
            echo "ACE is up and running!"
            break
          fi
          echo "Still waiting... ($i)"
          sleep 5
        done


    - name: Run Pytest tests
      #run: pytest
      run:  pytest tests/test_flow_conversion.py -v
      #run: pytest tests/ --maxfail=1 --disable-warnings -q

    - name: Stop and remove ACE container
```

```
    if: always()
    run: |
      docker stop ace-container
      docker rm ace-container

# ----------------------------------------------
# 🚀 Job 3: Deploy to AWS ECS
# ----------------------------------------------
deploy:
  needs: test-python
  runs-on: ubuntu-latest
  environment: production
  steps:
    - uses: actions/checkout@v5

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.ACCESS_KEY_AWS }}
        aws-secret-access-key: ${{ secrets.ACCESS_SECRET_KEY_AWS }}
        aws-region: us-east-1

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build, tag, and push image to Amazon ECR
      id: build-image
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        ECR_REPOSITORY: ace_http
        IMAGE_TAG: ${{ github.sha }}
      run: |
        docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
        echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTPUT

    - name: Fill in the new image ID in the Amazon ECS task definition
      id: task-def
      uses: aws-actions/amazon-ecs-render-task-definition@v1
      with:
        task-definition: ace_http_defination-revision2.json
        container-name: ace_http_container
        image: ${{ steps.build-image.outputs.image }}

    - name: Deploy ECS service
      uses: aws-actions/amazon-ecs-deploy-task-definition@v2
      with:
```

```
task-definition: ${{ steps.task-def.outputs.task-definition }}
service: ace_http_defination-service-zgjrpu93
cluster: ace_http_cluster
wait-for-service-stability: true
```

**********END**********

Step 18: Run and Verify the Pipeline

Once you've completed all setup steps and pushed your changes to the GitHub repository:

1. **Push all your changes to GitHub**
2. **Go to your GitHub repository**
    - Click on the **"Actions"** tab (top menu).
    - You'll see your pipeline running automatically (triggered by the push).
    - Wait until it reaches the step: **"🚀 Deploy ECS Service"**.
3. **Verify the Deployment on AWS**
    - Once deployment completes, open the AWS Console.
    - Navigate to **ECS → Clusters → ace_http_cluster → Tasks**.
    - Click on the **running task** to view its details.
4. **Get the Public IP**
    - In the task details page, locate the **Public IP** field under the networking section.
5. **Test Your Application**
    - Open your browser and access:
    - http://<PUBLIC_IP>:7600/

Example:

http://18.206.12.187:7600/

✅ **If you see your application or integration response, congratulations — your IBM ACE CI/CD pipeline is successfully deployed using GitHub Actions and AWS ECS!**

✅ **Outcome**

Once pushed to main, your pipeline will automatically:

1. Build and validate the BAR file
2. Run unit tests inside ACE container
3. Push the image to AWS ECR
4. Deploy the containerized ACE app to ECS automatically