**CI/CD Pipeline Documentation for fwf-ci-cd**

This document provides an overview of the CI/CD pipeline defined in the GitHub Actions workflow file, explaining its purpose, key components, and execution steps.

**Purpose**

The pipeline automates the **build**, **test**, and **deployment** process for the Flight Booking Aggregator project (fwf-ci-cd). It ensures that the application code is tested and deployed to an **Azure App Service** environment following each update in the main branch or manual workflow dispatch.

**Pipeline Triggers**

1. **On Push:**
   * Runs automatically when changes are pushed to the main branch.
2. **On Pull Request:**
   * Runs automatically for pull requests targeting the main branch.
3. **Manual Trigger (workflow\_dispatch):**
   * Allows users to manually trigger the workflow from the GitHub Actions tab.

**Environment Variables**

The pipeline relies on the following environment variables:

* **Azure Subscription & Resource Information:**
  + AZURE\_SUBSCRIPTION\_ID: Azure subscription ID for resource management.
  + AZURE\_RESOURCEGROUP\_NAME: Azure resource group name (fwf-rg).
* **Database Configuration:**
  + SQLSERVER\_NAME: SQL server name (fwf-db-srv).
  + DATABASE\_NAME: SQL database name (fwfdb).
  + SQLADMIN\_LOGIN: SQL Admin login name (stored in GitHub Secrets).
  + SQLADMIN\_PASS: SQL Admin password (stored in GitHub Secrets).
* **App Service Configuration:**
  + FWF\_APP\_SERVICE\_PLAN\_NAME: Name of the App Service Plan.
  + FWF\_FLIGHT\_SERVICE\_NAME: Azure App Service name for the Flight Search Service.
* **Code Paths:**
  + FWF\_SOLUTION\_PATH: Path to the solution file (repo/FWF/).
  + FWF\_FLIGHT\_SERVICE\_PATH: Path to the backend project.
  + FWF\_FLIGHT\_SERVICE\_PROJECT: Backend project name (FWF.FlightSearchService.App.csproj).
  + FWF\_FLIGHT\_SERVICE\_PUBLISH\_PROFILE: Publish profile for deployment (stored in GitHub Secrets).
* **DotNet Version:**
  + DOTNET\_VERSION: The version of .NET SDK to use (8.0.x).

**Pipeline Workflow**

**1. Code Checkout**

* **Step:** actions/checkout@v4
* **Purpose:** Clones the repository to the repo directory under $GITHUB\_WORKSPACE for the job to access the codebase.

**2. Setup .NET SDK**

* **Step:** actions/setup-dotnet@v1
* **Purpose:** Installs the specified .NET SDK (8.0.x) to prepare for build, test, and publish operations.

**3. Restore Dependencies**

* **Command:**

dotnet restore ${{env.FWF\_SOLUTION\_PATH}}/FWF.sln

* **Purpose:** Restores project dependencies based on the solution file.

**4. Build and Publish**

* **Commands:**

dotnet build ${{env.FWF\_SOLUTION\_PATH}}/FWF.sln --configuration Release --no-restore

dotnet publish ${{env.FWF\_FLIGHT\_SERVICE\_PATH}}/${{env.FWF\_FLIGHT\_SERVICE\_PROJECT}} -c Release -o ${{env.FWF\_FLIGHT\_SERVICE\_PATH}}/${{env.FWF\_FLIGHT\_SERVICE\_PROJECT}}/publish

* **Purpose:** Builds the solution and publishes the Flight Search Service project into a specified output directory.

**5. Run Unit Tests**

* **Command:**

dotnet test ${{env.FWF\_SOLUTION\_PATH}}/FWF.sln

* **Purpose:** Executes all unit tests to ensure code correctness before deployment.

**6. Azure Login**

* **Step:** azure/login@v1
* **Purpose:** Authenticates to Azure using a Service Principal credential stored in GitHub Secrets (FWF\_ENV\_SPN).

**7. Create Database Connection String**

* **Commands:**

CONN\_STR=$(az sql db show-connection-string --client ado.net --server ${{ env.SQLSERVER\_NAME }} --name ${{ env.DATABASE\_NAME }} -o tsv)

CONN\_STR=$(echo $CONN\_STR | sed "s/<username>/${{ env.SQLADMIN\_LOGIN }}/g")

CONN\_STR=$(echo $CONN\_STR | sed "s/<password>/${{ env.SQLADMIN\_PASS }}/g")

echo "SQL\_DB\_CONN\_STR=$CONN\_STR" >> $GITHUB\_ENV

* **Purpose:** Generates the ADO.NET connection string for the SQL database and adds it to the environment variables for later use.

**8. Update Database Schema**

* **Command:**

dotnet ef database update --project ${{env.FWF\_SOLUTION\_PATH}}/common/FWF.Dal/FWF.Dal.csproj --startup-project ${{env.FWF\_FLIGHT\_SERVICE\_PATH}}/${{env.FWF\_FLIGHT\_SERVICE\_PROJECT}}

* **Purpose:** Uses Entity Framework Core to update the database schema to the latest migration.

**9. Deploy to Azure App Service (Staging Slot)**

* **Step:** azure/webapps-deploy@v2
* **Purpose:** Deploys the published Flight Search Service project to the **staging slot** of the Azure App Service.

**10. Swap Slots to Production**

* **Commands:**

az webapp deployment slot swap --resource-group ${{ env.AZURE\_RESOURCEGROUP\_NAME }} --name ${{ env.FWF\_FLIGHT\_SERVICE\_NAME }} --slot staging --target-slot production

echo "Swap finished. Flight Search Service Application URL: https://$(az webapp show --resource-group ${{ env.AZURE\_RESOURCEGROUP\_NAME }} --name ${{ env.FWF\_FLIGHT\_SERVICE\_NAME }} --query hostNames[0] -o tsv)"

* **Purpose:** Promotes the staging deployment to the production environment by swapping slots.

**Deployment Strategy**

1. **Staging Slot Deployment:**  
   New versions are deployed to a staging slot to allow testing before going live.
2. **Slot Swap:**  
   Once validated, the staging environment is swapped with production to ensure zero-downtime deployment.