## horizontal line



Java React Demo - CICD & Infrastructure

**─**

Subhaijt Chatterji

Cloud DevOps Engineer

Azure | AWS

# Overview

The Project enables automated deployment of Java React Demo Project through Azure CICD Pipeline. The Project enables automated deployment of infrastructure required for the application in Azure

# Requirements - Prerequisites

1. Azure Subscription
2. Azure DevOps Subscription
3. Service Principal - ARM (Azure Resource Manager)
4. Service Principal - Azure Container Registry / Docker Registry
5. Visual Studio Code - IDE
6. Azure CLI
7. Terraform

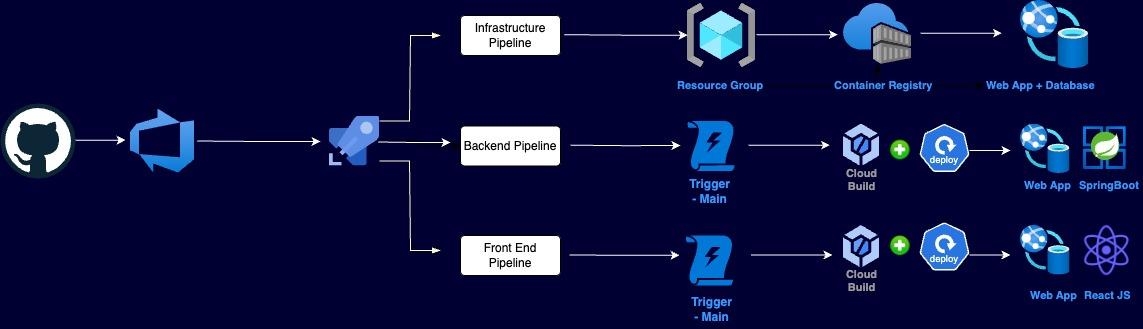
# Specifications

**Azure Resources Provisioned**:

* **Resource Group :** Created Resource Group to deploy the services is East US
* **Azure Container Registry (ACR)**: Created Azure Container Registry to deploy Frontend and Backend images.
* **Azure Web Apps for Containers**: Both the frontend and backend Docker images are deployed to separate Azure Web Apps + Database

Note : Azure Web Apps + Database is used to use cloud database instead of in-memory database as highlighted in requirements. (This incur charges) (Bonus)**\***

**Azure DevOps Architecture** :



**Azure DevOps Provisioned Pipelines**:

* Azure-pipelines-frontend.yml

| trigger:  branches:  include:  - main  pool:  vmImage: 'ubuntu-latest'  variables:  imageName: 'react-frontend'  azureSubscriptionServiceConnection: '98d927e6-6d13-4524-8723-0e7c858591e3'  devAppName: 'reactdemofrontend-dev'  prodAppName: 'reactdemofrontend-prod'  dockerRegistryServiceConnection: 'demodockerregistry'  acrLoginServer: 'javawebappcg.azurecr.io'  stages:  - stage: Build  displayName: 'Build Stage'  jobs:  - job: BuildAndPush  displayName: 'Build and Push Docker Image'  steps:  - task: NodeTool@0  inputs:  versionSpec: '18.x'  displayName: 'Install Node.js'  - script: |  npm install  npm run build  displayName: 'Build React App'  - task: Docker@2  inputs:  containerRegistry: '$(dockerRegistryServiceConnection)'  repository: '$(imageName)'  command: 'buildAndPush'  Dockerfile: '\*\*/Dockerfile'  tags: |  $(Build.BuildId)  displayName: 'Build and Push Docker Image'  - stage: Dev  displayName: 'Development Stage'  dependsOn: Build  jobs:  - job: DeployDev  displayName: 'Deploy to Development'  steps:  - task: AzureWebAppContainer@1  inputs:  azureSubscription: '$(azureSubscriptionServiceConnection)'  appName: '$(devAppName)'  containerRegistry: '$(dockerRegistryServiceConnection)'  imageName: '$(imageName)'  tag: '$(Build.BuildId)'  displayName: 'Deploy to Azure Web App - Dev'  - stage: Prod  displayName: 'Production Stage'  dependsOn: Dev  condition: succeeded()  jobs:  - job: DeployProd  displayName: 'Deploy to Production'  steps:  - task: AzureWebAppContainer@1  inputs:  azureSubscription: '$(azureSubscriptionServiceConnection)'  appName: '$(prodAppName)'  containerRegistry: '$(dockerRegistryServiceConnection)'  imageName: '$(imageName)'  tag: 'latest'  displayName: 'Deploy to Azure Web App - Prod' |
| --- |

* azure-pipelines-backend.yml

| trigger:  branches:  include:  - main  pool:  vmImage: 'ubuntu-latest'  variables:  imageName: 'java-backend'  azureSubscriptionServiceConnection: '98d927e6-6d13-4524-8723-0e7c858591e3'  devAppName: 'springbootdemobackend-dev'  prodAppName: 'springbootdemobackend-prod'  dockerRegistryServiceConnection: 'demodockerregistry'  acrLoginServer: 'javawebappcg.azurecr.io'  stages:  - stage: Build  displayName: 'Build Stage'  jobs:  - job: BuildAndPush  displayName: 'Build and Push Docker Image'  steps:  - task: Maven@3  inputs:  mavenPomFile: 'pom.xml'  mavenOptions: '-Xmx3072m'  javaHomeOption: 'JDKVersion'  jdkVersionOption: '1.11'  jdkArchitectureOption: 'x64'  publishJUnitResults: true  testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'  goals: 'package'  - task: Docker@2  inputs:  containerRegistry: '$(dockerRegistryServiceConnection)'  repository: '$(imageName)'  command: 'buildAndPush'  Dockerfile: '\*\*/Dockerfile'  tags: |  $(Build.BuildId)  displayName: 'Build and Push Docker Image'  - stage: Dev  displayName: 'Development Stage'  dependsOn: Build  jobs:  - job: DeployDev  displayName: 'Deploy to Development'  steps:  - task: AzureWebAppContainer@1  inputs:  azureSubscription: '$(azureSubscriptionServiceConnection)'  appName: '$(devAppName)'  containerRegistry: '$(dockerRegistryServiceConnection)'  imageName: '$(imageName)'  tag: '$(Build.BuildId)'  displayName: 'Deploy to Azure Web App - Dev'  - stage: Prod  displayName: 'Production Stage'  dependsOn: Dev  condition: succeeded()  jobs:  - job: DeployProd  displayName: 'Deploy to Production'  steps:  - task: AzureWebAppContainer@1  inputs:  azureSubscription: '$(azureSubscriptionServiceConnection)'  appName: '$(prodAppName)'  containerRegistry: '$(dockerRegistryServiceConnection)'  imageName: '$(imageName)'  tag: 'latest'  displayName: 'Deploy to Azure Web App - Prod' |
| --- |

* azure-pipelines-infrastructure.yml

| trigger:  branches:  include:  - main  pool:  vmImage: 'ubuntu-latest'  variables:  terraformWorkingDirectory: 'infrastructure'  azureResourceGroup: 'rg-java-react-app'  azureLocation: 'East Europe'  steps:  # Install Terraform  - task: TerraformInstaller@0  inputs:  terraformVersion: 'latest'  displayName: 'Install Terraform'  # Initialize Terraform without backend  - script: |  terraform init  workingDirectory: $(terraformWorkingDirectory)  displayName: 'Terraform Init'  # Terraform Plan  - script: |  terraform plan -var="resource\_group=$(azureResourceGroup)" \  -var="location=$(azureLocation)"  workingDirectory: $(terraformWorkingDirectory)  displayName: 'Terraform Plan'  # Terraform Apply  - script: |  terraform apply -auto-approve -var="resource\_group=$(azureResourceGroup)" \  -var="location=$(azureLocation)"  workingDirectory: $(terraformWorkingDirectory)  displayName: 'Terraform Apply'  # Output Results  - script: |  terraform output  workingDirectory: $(terraformWorkingDirectory)  displayName: 'Show Outputs' |
| --- |

* Azure-pipeline-infra-cleanup.yml (Bonus)**\***

| trigger:  branches:  include:  - main    pool:  vmImage: 'ubuntu-latest'  steps:  - script: |  terraform init  terraform destroy -auto-approve  displayName: 'Destroy Terraform Infrastructure' |
| --- |

**Modifications Made (Required)**

* Separate backend and frontend are created for different pipelines and containers (Bonus)**\***
* Minor Code modifications to fix errors in deployment
* Folder Structure   
    
  Java-React-Demo/

│

├── backend/

│ ├── src/

│ ├── pom.xml

│ └── Dockerfile

│

├── frontend/

│ ├── src/

│ ├── package.json

│ └── Dockerfile

│

├── infrastructure/

│ ├── main.tf

│ ├── variables.tf

│ └── outputs.tf

│

├── azure-pipelines-backend.yml

├── azure-pipelines-frontend.yml

└── azure-pipelines-infrastructure.yml  
 ├── azure-pipeline-infra-cleanup

* **Azure-pipeline-infra-cleanup (Bonus)\*** is created to provide cleanup/destroy functionality as part of the infra pipeline
* The delivered infrastructure is able to scale automatically.   
   resource "azurerm\_monitor\_autoscale\_setting" has been created for both backend and frontend, and can be found in code   
    
  ├── infrastructure/

├── main.tf

* **Azure Log Analytics Workspace** has been created to monitor the services deployed.
* Separate **Docker** files are created and are placed inside the frontend and backend folders to create separate images.
* Terraform Resources, variables, outputs and tfvars are created and are placed inside the infrastructure folder. To keep the document clean code is not copied here, but can be viewed from Git Repo.
* The Azure Services will be deleted once submitted as it’s on **pay-as-you-go** subscription and incur charges, the same is true for Azure DevOps pipelines and Agent pools incur charges on subscription