Full-Stack Microservices Deployment with CI/CD - Detailed Steps

# Step 1: Version Control & Project Setup

1. Create a GitHub Repository:

- Log in to GitHub and create a new repository for your project. Name it according to the application’s purpose (e.g., AdventureWorksMicroservices).  
 - Add a descriptive README.md to the repository with sections for project objectives, technology stack, setup instructions, and architecture overview.

2. Clone the Repository Locally:

- Open a terminal on your local machine and clone the repository:

git clone <your-repo-url>

- Navigate into the cloned directory:

cd AdventureWorksMicroservices

3. Initialize Git for Version Control:

git init

- Create .gitignore to exclude unnecessary files from version control (e.g., bin/, obj/):

bin/  
obj/  
\*.log  
\*.db

4. Commit Best Practices:

- Regularly commit changes as you develop, using meaningful commit messages:

git add .  
git commit -m 'Initialize project with README and .gitignore'

# Step 2: Database Integration

1. Set Up MS SQL Server:

- Download and install MS SQL Server.

- Open SQL Server Management Studio (SSMS) or a command-line interface.

2. Load Adventure Works Database:

- Download the Adventure Works database and attach it in SSMS.

- Use appropriate SQL commands to attach the database:

USE [master];  
CREATE DATABASE AdventureWorks  
ON (FILENAME = 'C:\Path\To\AdventureWorks.mdf')  
FOR ATTACH;

3. REST API Project in Visual Studio:

- Open Visual Studio and create a new ASP.NET Core Web API project.

- Configure the connection string in appsettings.json for the AdventureWorks database:

"ConnectionStrings": {  
 "DefaultConnection": "Server=localhost;Database=AdventureWorks;User Id=<username>;Password=<password>;"  
}

4. Create Controllers and CRUD Operations:

- Implement CRUD operations for selected tables using Entity Framework Core.

- Scaffold the database context and create API endpoints:

dotnet ef dbcontext scaffold "Server=localhost;Database=AdventureWorks;User Id=<username>;Password=<password>;" Microsoft.EntityFrameworkCore.SqlServer

# Step 3: Microservices Architecture with Docker

1. Set Up Docker:

- Download and install Docker Desktop. Ensure Docker is running.

2. Create Dockerfiles:

- Dockerfile for REST API: In the project folder, create a Dockerfile:

FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base  
WORKDIR /app  
EXPOSE 80  
COPY . .  
ENTRYPOINT ["dotnet", "YourApiProjectName.dll"]

- Dockerfile for MS SQL Server: Pull the MS SQL Server image if using Docker:

docker pull mcr.microsoft.com/mssql/server

3. Set Up Docker Compose:

- Create a docker-compose.yml file to manage the API and database containers:

version: '3.8'  
services:  
 sqlserver:  
 image: mcr.microsoft.com/mssql/server  
 environment:  
 SA\_PASSWORD: "YourStrong@Passw0rd"  
 ACCEPT\_EULA: "Y"  
 ports:  
 - "1433:1433"  
 api:  
 build: .  
 depends\_on:  
 - sqlserver  
 environment:  
 ConnectionStrings\_\_DefaultConnection: "Server=sqlserver;Database=AdventureWorks;User Id=sa;Password=YourStrong@Passw0rd;"  
 ports:  
 - "8080:80"

4. Build and Run Containers:

docker-compose up --build

# Step 4: Set Up CI/CD Pipeline with Jenkins

1. Install Jenkins:

- Install Jenkins and access it at http://localhost:8080.

2. Integrate GitHub with Jenkins:

- Configure GitHub webhooks to trigger Jenkins builds on each commit.

3. Jenkins Pipeline Setup:

- Create a Jenkinsfile with stages: Build, Docker Build, and Deploy to Staging:

pipeline {  
 agent any  
 stages {  
 stage('Build') {  
 steps {  
 script {  
 sh 'dotnet build YourApiProjectName.sln'  
 }  
 }  
 }  
 stage('Docker Build') {  
 steps {  
 script {  
 sh 'docker build -t your-api-image-name .'  
 }  
 }  
 }  
 stage('Deploy to Staging') {  
 steps {  
 script {  
 sh 'docker-compose up --build -d'  
 }  
 }  
 }  
 }  
}

# Step 5: Documentation

1. Document Architecture and Setup:

- Provide detailed explanations for each component.  
 - Include network or architecture diagrams.

2. API Documentation:

- Use Swagger for API documentation and provide interactive API specs.  
 - Include setup instructions in the README.md.

# Bonus Steps (Optional)

1. Implement API Versioning:  
 - Add versioning within controllers for backward compatibility.  
  
2. Deploy Jenkins to the Cloud:  
 - Configure Jenkins to deploy images to a cloud server (e.g., AWS or Azure) for production environments.