Deploying React Node Azure

If I have a React App as a frontend, Node.JS Express app as a backend and a Azure SQL Database that the backend integrates with, how can I deploy all this to a dedciated UAT enviroment using Microsoft Azure?

Deploying a React frontend, a Node.js Express backend, and an Azure SQL Database to a dedicated User Acceptance Testing (UAT) environment using Microsoft Azure involves several steps. Here's a high-level overview of the process:

1. Set Up Azure Resources

a. Create an Azure SQL Database

1. Go to the Azure Portal.
2. Create a new SQL Database.
   - Navigate to "Create a resource" > "Databases" > "SQL Database".
   - Fill in the necessary details like the resource group, database name, server name, and compute/storage options.
   - Set up the server admin login and password.
3. Configure the firewall settings to allow access from the Node.js backend.

b. Create an App Service for the Backend

1. In the Azure Portal, create a new App Service.
   - Navigate to "Create a resource" > "Compute" > "App Service".
   - Choose the resource group, app name, publish (Code), runtime stack (Node.js), and operating system.
   - Configure the App Service plan.

c. Create an App Service for the Frontend

1. Create another App Service following the same steps as for the backend, but this one will host the React app.

2. Deploy the Backend (Node.js Express App)

1. **Prepare the Backend for Deployment**
   - Ensure your `` `package.json` `` has the necessary scripts to start your app (e.g., `` `start` `` script).
   - Use environment variables for configuration (e.g., database connection strings).
2. **Deploy via Azure DevOps or GitHub Actions**
   - Set up a CI/CD pipeline using Azure DevOps or GitHub Actions.
   - Create a new pipeline and configure it to deploy the backend to the App Service.
   - Use the Azure App Service Deploy task in your pipeline.

```yaml
# Example GitHub Actions workflow file
name: Deploy Node.js App

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'
      - name: Install dependencies
        run: npm install
      - name: Build the app
        run: npm run build
      - name: Deploy to Azure Web App
        uses: azure/webapps-deploy@v2
        with:
          app-name: <your-backend-app-service-name>
          slot-name: 'production'
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
          package: .
```

3. Deploy the Frontend (React App)

1. **Prepare the Frontend for Deployment**
   - Build your React app using `**npm run build**`.
   - Ensure the build artifacts are ready for deployment (usually in the `**build**` directory).
2. **Deploy via Azure DevOps or GitHub Actions**
   - Similar to the backend, set up a CI/CD pipeline for the frontend.
   - Use the Azure App Service Deploy task in your pipeline.

```yaml
# Example GitHub Actions workflow file
name: Deploy React App

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'
      - name: Install dependencies
        run: npm install
      - name: Build the app
        run: npm run build
      - name: Deploy to Azure Web App
        uses: azure/webapps-deploy@v2
```

```
          with:
            app-name: <your-frontend-app-service-name>
            slot-name: 'production'
            publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
            package: ./build
```

4. Configuration and Environment Variables

- Set up necessary environment variables for both the backend and frontend apps in the App Services' configuration settings.
- For the backend, configure the database connection string.
- For the frontend, configure the backend API endpoint.

5. Testing and Validation

- Once deployed, test the UAT environment thoroughly.
- Ensure that the React frontend communicates properly with the Node.js backend.
- Verify the backend's connection to the Azure SQL Database.

Summary

1. Create necessary Azure resources (SQL Database, App Services).
2. Set up CI/CD pipelines for both backend and frontend using Azure DevOps or GitHub Actions.
3. Configure environment variables and deploy both applications.
4. Test the UAT environment to ensure everything works as expected.

Following these steps should help you deploy your full-stack application to a dedicated UAT environment on Microsoft Azure.

How would I pass in the host URL's into the React and Node.JS apps in this?

To pass the host URLs into your React and Node.js apps, you'll need to use environment variables. Here's how you can manage these variables for both development and production environments in your Azure setup:

1. Setting Environment Variables for the Node.js Backend

a. Local Development

Create a `.env` file in your Node.js project root directory and define your environment variables:

```ini
# .env
DB_HOST=<your-db-host>
DB_USER=<your-db-user>
```

```
DB_PASS=<your-db-password>
API_URL=<your-api-url>  # If your backend needs to call other APIs
```

In your Node.js application, you can access these variables using `process.env`:

javascript

```
// app.js or server.js
require('dotenv').config();

const express = require('express');
const app = express();
const dbHost = process.env.DB_HOST;
const dbUser = process.env.DB_USER;
const dbPass = process.env.DB_PASS;
const apiUrl = process.env.API_URL;

// Use these variables in your database connection and API calls
```

b. Azure App Service Configuration

1. Go to the Azure Portal.
2. Navigate to your Node.js App Service.
3. Under the “Settings” section, select “Configuration”.
4. Add the environment variables in the “Application settings” section.

2. Setting Environment Variables for the React Frontend

a. Local Development

Create a `.env` file in your React project root directory and define your environment variables. Note that React requires environment variables to start with `REACT_APP_`.

ini

```
# .env
REACT_APP_API_URL=http://localhost:5000  # Backend API URL
```

In your React application, you can access these variables using `process.env.REACT_APP_`:

javascript

```
// Example: src/api.js
const apiUrl = process.env.REACT_APP_API_URL;

export const fetchData = async () => {
  const response = await fetch(`${apiUrl}/data`);
  const data = await response.json();
  return data;
};
```

b. Azure App Service Configuration

1. Build your React app using `npm run build`.
2. During the deployment, ensure that the build process uses the correct environment variables. You can set environment variables directly in the Azure App Service:
    1. Navigate to your React App Service.
    2. Under the "Settings" section, select "Configuration".
    3. Add the environment variables in the "Application settings" section with the `REACT_APP_` prefix.

Alternatively, you can modify your CI/CD pipeline to inject environment variables during the build process.

3. Configuring CI/CD Pipelines to Use Environment Variables

a. GitHub Actions for Node.js Backend

Update your GitHub Actions workflow to include the environment variables:

```yaml
name: Deploy Node.js App

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'
      - name: Install dependencies
        run: npm install
      - name: Build the app
        run: npm run build
      - name: Deploy to Azure Web App
        uses: azure/webapps-deploy@v2
        with:
          app-name: <your-backend-app-service-name>
          slot-name: 'production'
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
          package: .
      - name: Set environment variables
        run: |
          az webapp config appsettings set --resource-group <your-resource-group> --name
<your-backend-app-service-name> --settings DB_HOST=${{ secrets.DB_HOST }} DB_USER=${{
secrets.DB_USER }} DB_PASS=${{ secrets.DB_PASS }} API_URL=${{ secrets.API_URL }}
```

b. GitHub Actions for React Frontend

Update your GitHub Actions workflow for the React app:

```yaml
name: Deploy React App
```

```
on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'
      - name: Install dependencies
        run: npm install
      - name: Build the app
        env:
          REACT_APP_API_URL: ${{ secrets.REACT_APP_API_URL }}
        run: npm run build
      - name: Deploy to Azure Web App
        uses: azure/webapps-deploy@v2
        with:
          app-name: <your-frontend-app-service-name>
          slot-name: 'production'
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
          package: ./build
      - name: Set environment variables
        run: |
          az webapp config appsettings set --resource-group <your-resource-group> --name
<your-frontend-app-service-name> --settings REACT_APP_API_URL=${{
secrets.REACT_APP_API_URL }}
```

Summary

1. Define environment variables locally in `.env` files for development.
2. Access these variables in your code using `process.env`.
3. Set environment variables in Azure App Services via the Azure Portal.
4. Configure your CI/CD pipelines to inject these variables during the build and deployment process.

This setup ensures that both your frontend and backend can dynamically access the necessary host URLs and other configuration settings, whether running locally or deployed on Azure.