Next.js AKS Deployment with DevSecOps Pipeline

This project demonstrates how to securely build, package, scan, and deploy a **Next.js** application to **Azure Kubernetes Service (AKS)** using a GitHub Actions pipeline. The deployment includes integration with **Azure Container Registry (ACR)**, **Helm charts**, **ingress controller**, and **security best practices**.

📌 Prerequisites

Before running the pipeline, ensure you have:

- An Azure subscription
- Azure CLI installed or access to Azure Cloud Shell
- A GitHub repository with the necessary secrets configured

To securely deploy to Azure from GitHub Actions, the following secrets are configured in the repository under
Settings → Secrets and variables → Actions:

- `AZURE_CREDENTIALS`
  A JSON object containing credentials for a service principal with access to Azure subscription.
  Used by the `azure/login@v1` action to authenticate to Azure.
  It includes:
  - `clientId`
  - `clientSecret`
  - `subscriptionId`
  - `tenantId`
- `ACR_USERNAME`
  The username for Azure Container Registry (ACR).
  This is typically the name of ACR instance (e.g., `nextjsbasicapp`).
- `ACR_PASSWORD`
  The password (secret) for the ACR.

🚀 CI/CD Pipeline Overview

The pipeline (`Deploy to AKS`) is triggered on each push to the `main` branch or manually via GitHub UI. It includes the following steps:

✅ Steps Explained

1. **Checkout Code**
   Clones the repository to the GitHub Actions runner.
2. **Tag Management**
   Fetches latest tags, increments patch version (`0.0.X`), and creates a new Git tag. This version is later injected into the Helm chart and Docker image.

3. **Node.js Setup & Install**
   Installs Node.js v22 and project dependencies via `npm install`.

4. **Security Checks**

   - `npm audit fix`: Automatically fixes known vulnerabilities.
   - `npm audit --audit-level=high`: Detects remaining high/critical issues.
   - `npm run lint`: Ensures code quality and coding standards.

5. **Build the App**
   Executes the Next.js build process (`npm run build`).

6. **Static Code Analysis**

   - **CodeQL Analysis**: Detects security flaws in JavaScript code with GitHub's built-in scanner.

7. **Build & Push Docker Image to ACR**

   - Uses `buildx` to build and tag the Docker image with `latest` and the version tag.
   - Pushes image to ACR.

8. **Inject Version into Helm**
   Updates the `Chart.yaml` and `values.yaml` with the correct version and image tag.

9. **Package & Push Helm Chart to ACR**

   - Packages Helm chart.
   - Logs in to ACR.
   - Pushes the chart to the OCI registry in ACR.

10. **Helm Deployment to AKS**

- Uses Helm to deploy the application to AKS using the updated chart.
- Waits for pod readiness and validates deployment.

Great point — using `nip.io` for dynamic DNS is a smart move in development or testing environments. Here's your enhanced `Ingress` section, now including:

- SSL/TLS support
- Access via the `nip.io` dynamic DNS domain
- How it works and why it's helpful

🌐 Exposing the Application with Ingress and SSL

The application is exposed via an Ingress resource that provisions an external **public IP** with support for both HTTP and HTTPS.

✅ Ingress Overview

After deployment, the Azure Load Balancer public IP acn be obtained with

```
kubectl get ingress
```

Example output:

```
NAME               CLASS   HOSTS   ADDRESS          PORTS     AGE
nextjsbasic-helm   nginx   *       128.203.114.45   80, 443   8m58s
```

- `ADDRESS`: `128.203.114.45` — Azure Load Balancer public IP
- `PORTS`: Supports both HTTP (80) and HTTPS (443)

🌍 Accessing the App

The app is publicly available via the IP address:

```
https://128.203.114.45
```

Or, more conveniently, via a dynamic DNS domain using nip.io:

```
https://nextjsbasicapp.128.203.114.45.nip.io/
```

This works automatically because `*.nip.io` resolves the domain to the IP embedded in it (`128.203.114.45`).
No DNS configuration is required — great for demos and test environments.
