Design Document:

Nginx Hello World Application with GitHub Pipeline and Kubernetes Deployment

1. Overview

This document outlines the design for deploying a simple Nginx "Hello World" application using a GitHub pipeline, Docker, and Kubernetes. The solution will:

1. Build and package an Nginx-based application.
2. Automatically update and push a Docker image to Docker Hub on code changes.
3. Deploy the containerized application as a Kubernetes Pod with a Deployment and Service.

2. Architecture

Components:

1. **GitHub Repository**: Stores the application source code, Dockerfile, and Kubernetes manifests.
2. **GitHub Actions Pipeline**: Automates building, testing, and pushing the Docker image.
3. **Docker Hub**: Stores the built container images.
4. **Kubernetes Cluster**: Deploys and manages the application.

Flow:

- Developer pushes code to GitHub.
- GitHub Actions builds a new Docker image with an updated HTML file.
- The image is pushed to Docker Hub.
- Kubernetes Deployment fetches the new image and deploys it.

3. Implementation Details

3.1 Nginx Application

A simple HTML file will be served using an Nginx container.

File: index.html

```
<html>
 <head>
  <title>Hello World</title>
 </head>
 <body>
  <h1>Hello, World from Nginx!</h1>
 </body>
</html>
```

File: Dockerfile

FROM nginx:latest

COPY index.html /usr/share/nginx/html/index.html

3.2 GitHub Actions Pipeline

The pipeline will:

1. Build the Docker image.
2. Tag the image with a version or commit hash.
3. Push the image to Docker Hub.

File: .github/workflows/deploy.yml

```
name: Build and Deploy

on:
  push:
    branches:
      - main
env:
  DOCKER_IMAGE: test/nginx-hello-world
jobs:
  build:
    runs-on: <runner-tag-name>
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3
      - name: Log in to Docker Hub
        run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{
secrets.DOCKER_USERNAME }}" --password-stdin


      - name: Build and Tag Docker Image
        run: |
          docker build -t $DOCKER_IMAGE:${{ github.sha }} .
          docker tag $DOCKER_IMAGE:${{ github.sha }} $DOCKER_IMAGE:latest


      - name: Push Docker Image
        run: |
          docker push $DOCKER_IMAGE:${{ github.sha }}
          docker push $DOCKER_IMAGE:latest
```

```
  - name: Deploy to Kubernetes
    run: |
      kubectl create namespace nginx-k8s --dry-run=client -o yaml | kubectl apply -f -
      kubectl apply -f deployment.yaml -n nginx-k8s
      kubectl apply -f service.yaml -n nginx-k8s

  - name: Verify Deployment
    run: kubectl get pods -n nginx-k8s && kubectl get svc -n nginx-k8s
```

3.3 Kubernetes Deployment

A Kubernetes Deployment and Service will be created to expose the application.

File: deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-hello-world
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx-hello-world
 template:
  metadata:
   labels:
    app: nginx-hello-world
  spec:
   containers:
    - name: nginx
```

```
      image: yourdockerhubusername/nginx-hello-world:latest
      ports:
       - containerPort: 80
```

File: service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx-hello-world
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort
```

3.4 Deployment Steps

1. Apply the Kubernetes manifests:
2. kubectl apply -f deployment.yaml
3. kubectl apply -f service.yaml
4. Get the service URL:

   kubectl get svc nginx-service
5. Access the application using the external IP of the LoadBalancer.

4. Security Considerations

- Store Docker credentials as GitHub Secrets.
- Use a private Docker repository required.

5. Conclusion

This document outlines a simple and scalable way to deploy an Nginx "Hello World" application using GitHub Actions, Docker Hub, and Kubernetes. The design ensures automation, scalability, and maintainability.